REMARKS

The Office Action in the above-identified application has been carefully considered and this amendment has been presented to place this application in condition for allowance. Accordingly, reexamination and reconsideration of this application are respectfully requested.

Claims 1, 4-14, 16, and 19-29 are in the present application. It is submitted that these claims were patentably distinct over the prior art cited by the Examiner, and that these claims were in full compliance with the requirements of 35 U.S.C. § 112. The changes to the claims, as presented herein, are not made for the purpose of patentability within the meaning of 35 U.S.C. sections 101, 102, 103 or 112. Rather, these changes are made simply for clarification and to round out the scope of protection to which Applicant is entitled.

In response to the Examiner's request, enclosed is a copy of the article by S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," Int. J. Satellite Commun., Vol. 16, pp. 23-46, Jan-Feb 1998.

Claims 1, 4-14, 16, and 19-29 were rejected under 35 U.S.C. § 112, second paragraph, as being incomplete for omitting essential structural cooperative relationships between elements. In response, Applicant has deleted the offending limitation "the input is digital information encoded as convolutional codes." Accordingly, Applicant believes this rejection has been overcome.

Claims 1, 4-14, 16, and 19-29 were rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. Specifically, the Examiner takes issue with "a linear approximation means" implementing a mathematical algorithm. (Office Action pages 2-3) Although Applicant still disagrees with the Examiner's position and reasserts the prior arguments, Applicant has amended independent claims 1 and 16 to directly tie the limitations to specific hardware elements in the decoder which perform the novel correction term computation portion of a log likelihood calculation used in decoding the encoded input signal. Specifically, the present claims are directed to "a decoder for decoding digital information from an encoded input signal received over a communication channel." (Claim 1, Claim 16 contains similar limitations) This limitation relates directly to the hardware system elements (decoder 3 and memoryless communication channel 2) shown in Figure 6 and described on page 20 of the specification. Moreover, the claims now recite "an absolute value computation circuit for calculating a variable based on the absolute value of said received value." (Claim 1, Claim 16 contains similar limitations) As shown in Figure 14, the absolute value computation circuit 67 and the linear approximation circuit 68 (i.e. the claimed linear approximation means) make up the correction term computation circuit 65 which occurs within several different components in the decoder 3. Accordingly, the recited limitations now clearly describe functions performed by discrete hardware elements within the decoder. For this reason, Applicant believes the present invention is clearly statutory subject matter as the invention is hardware/software which practically applies a mathematical algorithm to produce a tangible result (decoding digital information from an encoded input signal).

In rebuttal to the Examiner's Response to Arguments, the Examiner states that unlike digital data, a digital signal requires hardware. Applicant does not understand the distinction being made, since presumably both are comprised of bits (1s and 0s) which cannot exist without a hardware construct. Secondly, the valid example provided in the MPEP as a statutory process does not recite any hardware. Thirdly, the Examiner states that data structures are non-statutory. However, an accompanying mention of a hardware element (e.g. a memory) has been found to make a data structure statutory. Presumably, the present invention's "decoder" would suffice as such a hardware element to make the decoded data statutory.

Therefore, Applicant believes the amended claims constitute statutory subject matter and respectfully requests this rejection be withdrawn.

In view of the foregoing amendment and remarks, it is respectfully submitted that the application as now presented is in condition for allowance. Early and favorable reconsideration of the application are respectfully requested.

No fees are deemed to be required for the filing of this amendment, but if such are, the Examiner is hereby authorized to charge any insufficient fees or credit any overpayment associated with the above-identified application to Deposit Account No. 50-0320.

If any issues remain, or if the Examiner has any further suggestions, he/she is invited to call the undersigned at the telephone number provided below. The Examiner's consideration of this matter is gratefully acknowledged.

Respectfully submitted,
FROMMER LAWRENCE & HAUG LLP

By: [illegible]
Darren M. Simon
Reg. No. 47,946
(212) 588-0800

INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS
Int. J. Satell. Commun., 16, 23–46 (1998)

IMPLEMENTATION AND PERFORMANCE OF A TURBO/MAP DECODER

STEVEN S. PIETROBON*
Small World Communications, 6 First Avenue, Payneham South, SA 5070, Australia

SUMMARY

The implementation and performance of a turbo/MAP decoder are described. A serial block MAP decoder operating in the logarithm domain is used to obtain a very-high-performance turbo decoder. Programmable gate arrays and EPROMs allow the decoder to be programmed for almost any code from four to 512 states, rate 1/3 to rate 1/7 (higher rates are achieved with puncturing) and interleaver block sizes to 65,536 bits. Seven decoding stages were implemented in parallel. For rate 1/3 and 1/7 16-state codes with an interleaver size of 65,536 bits and operating at up to 356 kbit/s the codec achieved an E_b/N_0 of 0·32 and −0·30 dB respectively for a BER of 10^{-5}. BERs down to 10^{-7} were also achieved for a small increase in E_b/N_0. An efficient implementation of a continuous MAP decoder is also presented, along with a synchronization technique for turbo decoders. © 1998 John Wiley & Sons, Ltd.

KEY WORDS: turbo coding; MAP decoding; synchronization

1. INTRODUCTION

Error control coding aims to correct errors caused by noise and interference in a digital communications scheme. For power-limited schemes the ratio of energy per bit to single-sided noise density (E_b/N_0) is desired to be as low as possible. Good examples of this are satellite and space communications, where fairly low bandwidth efficiencies (K, in bits transmitted per signalling interval or bit/sym) of 0·1–2 bit/sym are used.

Coding adds redundancy using special codes. A decoder will then use this redundant information to correct as many errors as possible. Shannon[1] showed that for an additive white Gaussian noise channel the smallest E_b/N_0 that can be obtained for reliable transmission is

$$\frac{E_b}{N_0} \geq \frac{2^K - 1}{K} \tag{1}$$

As K approaches zero or the required bandwidth approaches infinity, the smallest value of E_b/N_0 is ln2 or approximately −1·59 dB. A typical scheme such as uncoded quadrature phase shift keying (QPSK) with $K = 2$ bits/sym requires an E_b/N_0 of 9·6 dB for a bit error ratio (BER) of 10^{-5}. Figure 1 plots K versus E_b/N_0, showing the Shannon capacity curve from (1) and the capacity curve when QPSK modulation is used.[2] Also plotted are the performances of uncoded QPSK and some other coding schemes at a BER less than or equal to 10^{-5}. Note that the Voyager and Galileo schemes use binary phase shift keying (BPSK), which would result in their bandwidth efficiencies being reduced by half. However, since Gray-mapped QPSK can achieve the same performance as BPSK with twice the bandwidth efficiency, we plot the Voyager and Galileo schemes using QPSK.

The industry standard code for satellite communications is a rate 1/2, 64-state, non-systematic convolutional code[3] with a soft-decision Viterbi decoder. This code can achieve an E_b/N_0 of 4·2 dB at a BER of 10^{-5}, giving a 5·4 dB coding gain. In practice, the use of 3 bit soft decisions and other quantization effects in the decoder results in a 0·2 dB performance loss. Another 0·2 dB may be lost owing to the use of differential encoding to resolve 180° phase ambiguities in the QPSK signal set.

To obtain better performance, the standard code has been concatenated with a Reed–Solomon (RS) outer code. The most famous example is the (255,223) GF(2^8) RS code with depth eight interleaving used on the Voyager space probes.[3,4] This scheme can achieve an E_b/N_0 of 2·53 dB at a BER of 10^{-6} and $K = 0·875$ bit/sym. A more advanced scheme uses a rate 1/4, 8192-state, non-systematic convolutional inner code and a time-varying, depth eight GF(2^8) RS outer code with redundancy profile (94,10,30,10,60,10,30,10), giving a (255,223·25) code on average.[5] Four stages of iterative decoding are used to obtain an E_b/N_0 of 0·58 dB at a BER of 10^{-7} and $K = 0·438$ bit/sym. This is the most powerful code currently in use and is 1·5 dB from Shannon capacity at $K = 0·438$ bit/sym.

* Correspondence to: S. S. Pietrobon, Small World Communications, 6 First Avenue, Payneham South, SA 5070, Australia.
Contract/grant sponsor: Australian Research Council
Contract/grant sponsor: University of South Australia

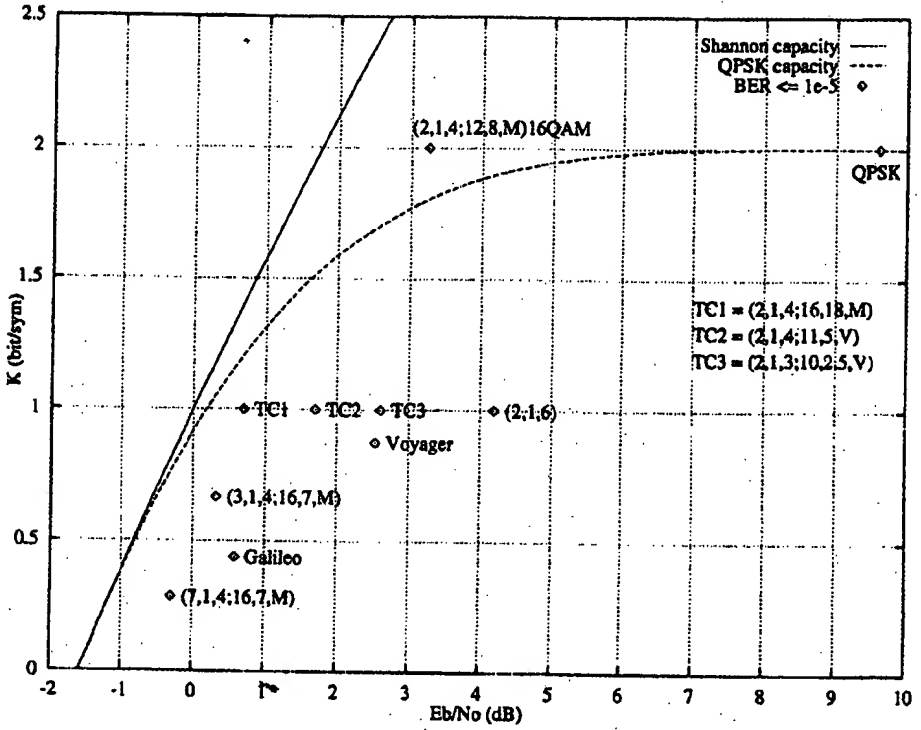

Figure 1. Shannon and QPSK capacity

In 1993, Berrou *et al.* published a paper describing a new coding scheme called 'turbo-codes'.[6] A rate 1/2 code is described that achieved the amazing performance of $E_b/N_0 = 0{\cdot}7$ dB at a BER of 10^{-5}. This is only 0·7 and 0·5 dB from Shannon and QPSK capacity respectively at $K = 1$ bit/sym. The encoder consists of two parallel concatenated systematic convolutional encoders separated by a random interleaver. The code in Reference 6 used two punctured 16-state codes and a 65,536 bit interleaver. This code is shown as TC1 in Figure 1.

Decoding is performed iteratively. Each systematic code is decoded using a soft-in soft-out (SISO) decoder. The output of the first decoder feeds into the second decoder to form one turbo decoder iteration. Eighteen iterations were performed in Reference 6. The SISO decoder used in Reference 6 is a modification of the maximum *a posteriori* (MAP) decoding algorithm.[7] A similar iterative decoding technique is described by Gallager in his 1962 paper on low-density parity check codes.[8] Each parity bit together with its associated checked information bits is treated as a single $(k+1,k)$ block code. A simple soft-output MAP decoding algorithm is used for each of the parity bits. A second SISO MAP decoder is then used to decode the information bit associated with the j parity check bits. The process then repeats.

The MAP algorithm finds the most likely information bit to have been transmitted in a coded sequence. This is unlike the Viterbi algorithm,[9] which finds the most likely sequence to have been transmitted. When the decoded BER is small, there is a negligible error performance difference between the MAP and Viterbi algorithms. Since the MAP algorithm is considerably more complex than the Viterbi algorithm, it has thus been largely ignored. However, at low E_b/N_0 and high BERs, MAP can outperform soft-output Viterbi by 0·5 dB or more. For turbo codes this is very important, since the output BERs from the first stages of iterative decoding can be very high. Thus any improvement that can be obtained at these high BERs will directly result in performance increases.

A practical application of turbo/MAP decoders for satellite communications is given in Reference 10. Various turbo coding schemes were investigated to achieve 2 bit/sym bandwidth efficiency over a high-speed mobile satellite link. Using a rate 1/2 turbo code with a 4000 bit block size, eight iterations and 16QAM modulation, an E_b/N_0 of 3·25 dB could be achieved over an ideal AWGN channel for a BER of 10^{-5}. This is only 1·5 dB from Shannon capacity at 2 bit/sym. This code is shown as (2,1,4;12,8,M)16QAM in Figure 1. The notation $(n,k,v;m,I,j)$ is used to describe a turbo code, with n the number of coded bits, k the number of information bits (the code rate $R = k/n$), v the memory of individual encoders (the number of states is equal to 2^v), $m = \log_2 N_I$ (where N_I is the interleaver size), I the number of decoder iterations and $j = \text{M}$ or V indicating MAP or SOVA decoders respectively.

The MAP algorithm does not have to be used in a turbo decoder. The simpler soft-output Viterbi algorithm (SOVA)[11-14] can also be used. However, since the output of SOVA does not provide as good a statistic as MAP and makes more errors, degradations of about 0·8 dB can be expected in overall performance.[15] SOVA has been used in Reference 16 to implement a turbo decoder on a chip. Two punctured eight-state codes were used to obtain a rate 1/2 turbo code which achieves an E_b/N_0 of 2·6 dB at a BER of 10^{-5}. This is almost the same performance as the more complicated Voyager code! The interleaver size is 1024 bits and 2·5 iterations (five SOVA decoders) were used. The code is shown as TC3 in Figure 1.

A single iteration on a chip was implemented in Reference 17. This chip consists of two 16-state SOVA decoders and a 2048 bit interleaver/deinterleaver. A performance of $E_b/N_0 = 1{\cdot}7$ dB at a BER of 10^{-5} is achieved after five iterations for a rate 1/2 turbo code. This is TC2 in Figure 1.

An obstacle to implementing the code in Reference 6 is its sheer complexity. The MAP algorithm described in Reference 6 is very complex and is not very amenable to hardware implementation. By operating the algorithm in the logarithm domain,[18,19] the complexity can be greatly reduced. In this paper we describe the implementation and performance of a turbo/MAP decoder which can implement the code in Reference 5. The MAP decoder was designed to be very flexible and can be programmed to operate from four to 512 states and from rate 1/2 to rate 1/4. Punctured codes can also be implemented. The decoding speed ranges from 17·7 kbit/s for 512 states to 624·4 kbit/s for four states. A 16-state code operates at 356·8 kbit/s.

We first give the derivation of the MAP, log-MAP and sub-MAP decoding algorithms. This is followed by a description of an implementation of the log-MAP algorithm. We then describe the iterative turbo decoding algorithm and give a description of its implementation. Actual performance curves of the decoder are presented, followed by some comments on continuous decoding and synchronization.

2. THE MAP, LOG-MAP AND SUB-MAP ALGORITHMS

We present here a full derivation of the MAP decoding algorithm for systematic convolutional codes on an additive white Gaussian noise (AWGN) channel. The derivation is similar to that in Reference 18, with the final presentation of the algorithm being slightly simpler than the traditional presentations.

2.1. *The MAP Decoding Algorithm*

The origin of the MAP algorithm belongs to Chang and Hancock,[20] who developed it to minimize the symbol (or bit) error probability for an intersymbol interference (ISI) channel. Simultaneously, Bahl *et al.*[21] and McAdam *et al.*[22] developed the algorithm for use on coded channels. The MAP algorithm that was presented in Reference 6 for systematic convolutional codes is very complicated. A simplified version of this algorithm is given in Reference 18. However, Berrou and Glavieux changed over to the more traditional presentation[15,23] of the algorithm in Reference 24.

For an encoder with υ memory cells we define the encoder state at time k, S_k, as a ν-tuple, depending only on the output of each delay element. The information bit at time k, d_k, is associated with the transition from time k to time $k+1$ and will change the encoder state from S_k to S_{k+1}. Also suppose that the information bit sequence $\{d_k\}$ is made up of $N-\nu$ independent bits d_k, taking values zero and one with *a priori probability* (APrP) ζ_k^0 and ζ_k^1 respectively ($\zeta_k^0+\zeta_k^1=1$). We let the encoder initial state S_1 be equal to zero. The last ν information bits ($d_{N-\nu+1}$ to d_N) are set to values that will force the state to zero at time $N+1$ (i.e. $S_{N+1}=0$). This will slightly reduce the rate of the encoder.

We consider a rate 1/2 systematic feedback encoder whose outputs at time k are the uncoded data bit d_k and the coded bit c_k. These outputs are modulated with a BPSK or QPSK modulator and sent through an AWGN channel. At the receiver end we define the received sequence

$$R_1^N = (R_1,\ldots,R_k,\ldots,R_N) \tag{2}$$

where $R_k = (x_k, y_k)$ is the received symbol at time k; x_k and y_k are defined as

$$x_k = (2d_k - 1) + p_k \tag{3}$$

$$y_k = (2c_k - 1) + q_k \tag{4}$$

with p_k and q_k being two independent, normally distributed random variables with variance σ^2. We define the *likelihood ratio* λ_k associated with each decoded bit d_k as

$$\lambda_k = \frac{\Pr(d_k = 0|R_1^N)}{\Pr(d_k = 1|R_1^N)} \tag{5}$$

where $\Pr(d_k = i|R_1^N)$, $i = 0,1$, is the *a posteriori probability* (APoP) of the data bit d_k. The APoP of a decoded data bit d_k can be derived from the joint probability defined by

$$\lambda_k^{i,m} = \Pr(d_k = i, S_k = m|R_1^N) \tag{6}$$

and thus the APoP of a decoded data bit d_k is equal to

$$\Pr(d_k = i|R_1^N) = \sum_m \lambda_k^{i,m} \tag{7}$$

where $i = 0,1$ and the summation is over all 2^ν encoder states. From (5) and (7) the λ_k associated with a decoded bit d_k can be written as

$$\lambda_k = \frac{\sum_m \lambda_k^{0,m}}{\sum_m \lambda_k^{1,m}} \tag{8}$$

The decoder can make a decision by comparing λ_k with a threshold equal to one:

$$\hat{d}_k = \begin{cases} 0, & \lambda_k \geq 1 \\ 1, & \lambda_k < 1 \end{cases} \tag{9}$$

Using Bayes' rule, the joint probability from (6) can be rewritten as

$$\begin{aligned} \lambda_k^{i,m} &= \Pr(d_k = i, S_k = m, R_1^N)/\Pr(R_1^N) \\ &= \Pr(R_1^{k-1} | d_k = i, S_k = m, R_k^N) \\ &\times \Pr(R_{k+1}^N | d_k = i, S_k = m, R_k) \\ &\times \Pr(d_k = i, S_k = m, R_k)/\Pr(R_1^N) \end{aligned} \tag{10}$$

We have

$$\begin{aligned} &\Pr(R_1^{k-1} | d_k = i, S_k = m, R_k^N) \\ &\quad = \Pr(R_1^{k-1} | S_k = m) = \alpha_k^m \end{aligned} \tag{11}$$

since the assumption that $S_k = m$ implies that events before time k are not influenced by observations after time k. We define α_k^m as the forward state metric at time k and state m. Similarly, we have

$$\begin{aligned} &\Pr(R_{k+1}^N | d_k = i, S_k = m, R_k) \\ &= \Pr(R_{k+1}^N | S_{k+1} = f(i,m)) = \beta_{k+1}^{f(i,m)} \end{aligned} \tag{12}$$

where $f(i,m)$ is the next state given an input i and state m. We define β_k^m as the reverse state metric at time k and state m. We define the branch metric as

$$\delta_k^{i,m} = \Pr(d_k = i, S_k = m, R_k) \tag{13}$$

Substituting (11)–(13) into (10), we obtain

$$\lambda_k^{i,m} = \alpha_k^m \delta_k^{i,m} \beta_{k+1}^{f(i,m)}/\Pr(R_1^N) \tag{14}$$

This result can be used to evaluate (8) as

$$\lambda_k = \frac{\sum_m \alpha_k^m \delta_k^{0,m} \beta_{k+1}^{f(0,m)}}{\sum_m \alpha_k^m \delta_k^{1,m} \beta_{k+1}^{f(1,m)}}$$

where the summations are over all 2^ν states. The usual expression for (15) involves a double summation in both the numerator and denominator.

We want to show that (11) can be recursively calculated. We can express (11) as

$$\begin{aligned} \alpha_k^m &= \Pr(R_1^{k-1} | S_k = m) \\ &= \sum_{m'} \sum_{j=0}^{1} \Pr(d_{k-1} = j, S_{k-1} = m', R_1^{k-1} | S_k = m) \\ &= \sum_{m'} \sum_{j=0}^{1} \Pr(R_1^{k-2} | S_k = m, d_{k-1} = j, S_{k-1} = m', R_{k-1}) \\ &\times \Pr(d_{k-1} = j, S_{k-1} = m', R_{k-1} | S_k = m) \\ &= \sum_{j=0}^{1} \Pr(R_1^{k-2} | S_{k-1} = b(j,m)) \\ &\times \Pr(d_{k-1} = j, S_{k-1} = b(j,m), R_{k-1}) \\ &= \sum_{j=0}^{1} \alpha_{k-1}^{b(j,m)} \delta_{k-1}^{j,b(j,m)} \end{aligned} \tag{16}$$

where the first summation is from $m' = 0$ to $2^\nu - 1$ and $b(j,m)$ is the state going backwards in time from state m on the previous branch corresponding to input j. In a similar way we can recursively calculate the probability β_k^m from the probability β_{k+1}^m. Note that this is possible only after the whole block of data is received. Relation (12) becomes

$$\begin{aligned} \beta_k^m &= \Pr(R_k^N | S_k = m) \\ &= \sum_{m'} \sum_{j=0}^{1} \Pr(d_k = j, S_{k+1} = m', R_k^N | S_k = m) \\ &= \sum_{m'} \sum_{j=0}^{1} \Pr(R_{k+1}^N | S_k = m, d_k = j, S_{k+1} = m', R_k) \\ &\times \Pr(d_k = j, S_{k+1} = m', R_k | S_k = m) \\ &= \sum_{j=0}^{1} \Pr(R_{k+1}^N | S_{k+1} = f(j,m)) \\ &\times \Pr(d_k = j, S_k = m, R_k) \\ &= \sum_{j=0}^{1} \delta_k^{j,m} \beta_{k+1}^{f(j,m)} \end{aligned} \tag{17}$$

Figure 2 gives a graphical illustration of the calculation of α_k^m and β_k^m. It is very similar to the architecture of the Viterbi algorithm. Where we add the branch metric to the state metric in the Viterbi algorithm, we multiply in the MAP algorithm. Where we find the minimum of the path metrics in the Viterbi algorithm, we add in the MAP algorithm. Thus the add–compare–select (ACS) operation in the Viterbi algorithm becomes the multiply–add (MA) operation in the MAP algorithm.

The MAP algorithm works by first calculating the α_k^ms in the forward direction and storing the results. The β_k^ms are then calculated in the reverse direction. An important observation is that the $\delta_k^{j,m} \beta_{k+1}^{f(j,m)}$ term in (17) is also used in the calculation of λ_k in (15). Thus, while the β_k^ms are being calculated, λ_k should be calculated at the same time, reusing the $\delta_k^{j,m} \beta_{k+1}^{f(j,m)}$ terms to minimize the number of computations. If the block starts in state zero, then we

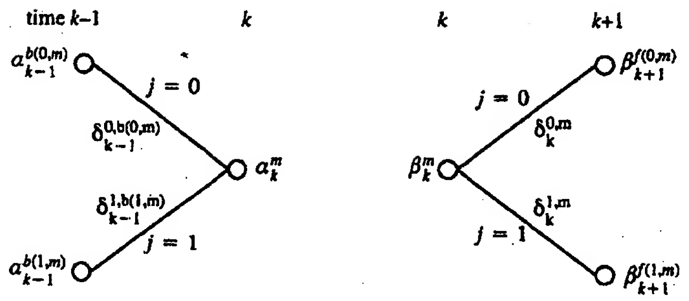

Figure 2. Graphical representation of calculation of α_k^m and β_k^m

initialize $\alpha_1^0=1$ and $\alpha_1^m=0$ for $m\neq 0$. A similar initialization is performed for β_{N+1}^m if the block ends in state zero. If the block ends in an unknown state (which occurs when there is no termination), then $\beta_{N+1}^m=1$ for all m.

The branch metric $\delta_k^{i,m}$ can be determined from the transition probability of the discrete memoryless channel and the APrP. From (13) and using Bayes' rule, we have

$$\begin{aligned}\delta_k^{i,m} &= \Pr(d_k=i,S_k=m,R_k)\\ &= \Pr(R_k|d_k=i,S_k=m)\\ &\quad\times \Pr(S_k=m|d_k=i)\Pr(d_k=i)\\ &= \Pr(x_k|d_k=i,S_k=m)\\ &\quad\times \Pr(y_k|d_k=i,S_k=m)\zeta_k^i/2^\nu \end{aligned} \tag{18}$$

since p_k and q_k are independent, the current state is independent of the current input and can be in any of the 2^ν states, and $\zeta_k^i=\Pr(d_k=i)$ by definition. For and AWGN channel with zero mean and variance σ^2, (18) becomes

$$\begin{aligned}\delta_k^{i,m} &= \frac{\zeta_k^i}{2^\nu\sqrt{2\pi}\,\sigma}\exp\left(\frac{1}{2\sigma^2}[x_k-(2i-1)]^2\right)\mathrm{d}x_k\\ &\quad\times\frac{1}{\sqrt{2\pi}\,\sigma}\exp\left(-\frac{1}{2\sigma^2}[y_k-(2c^{i,m}-1)]^2\right)\mathrm{d}y_k\\ &= \kappa_k\zeta_k^i\exp[L_c(x_ki+y_kc^{i,m})]\end{aligned} \tag{19}$$

where κ_k is a constant, $\mathrm{d}x_k$ and $\mathrm{d}y_k$ are the differentials of x_k and y_k, $L_c=2/\sigma^2$ and $c^{i,m}$ is the coded bit given $d_k=i$ and $S_k=m$. Since the constant κ_k in (19) does not affect λ_k in (15), we can normally ignore κ_k. In practice, though, when calculating the forward and reverse state metrics, we let κ_k be equal to the inverse of the largest previous state metric. This normalizes the new state metrics and ensures that the state metrics do not under- or overflow.

If we substitute (19) into (15), we obtain

$$\begin{aligned}\lambda_k &= \frac{\zeta_k^0}{\zeta_k^1}\exp(-L_cx_k)\\ &\quad\times\frac{\sum_m \alpha_k^m\exp(L_cy_kc^{0,m})\beta_{k+1}^{f(0,m)}}{\sum_m \alpha_k^m\exp(L_cy_kc^{1,m})\beta_{k+1}^{f(1,m)}}\\ &= \zeta_k\exp(-L_cx_k)\zeta'_k\end{aligned} \tag{20}$$

where $\zeta_k=\zeta_k^0/\zeta_k^1$ is the input APrP ratio and ζ'_k is the output *extrinsic information*. One can think of ζ'_k as a correction term that changes the input information so as to minimize the probability of decoding error. This extrinsic information is very important in turbo decoding as it allows the corrections terms to be passed from one decoder to the next.

2.2. *The log-MAP Decoding Algorithm*

To minimize the decoding complexity, we would like to eliminate the multiply operations required by the MAP algorithm. This can be achieved by taking the logarithm (or negative logarithm) of the algorithm. Again, this technique was first used for the ISI channel.[25,26] It was later applied to the coding channel in References 18, 19 and 27–29.

Taking the negative logarithm, the multiplications in the algorithm are converted to additions. Adders are much easier to implement than multipliers. However, the additions are converted to the E operand defined below:

$$\begin{aligned}a\,\mathrm{E}\,b &\equiv -\log_\epsilon(\epsilon^{-a}+\epsilon^{-b})\\ &= \min(a,b)-\log_\epsilon(1+\epsilon^{-|a-b|})\end{aligned} \tag{21}$$

The functions $\min(a,b)$ and $|a-b|$ can be easily determined using subtraction and multiplexer circuits. However, the function

$$f(z)=\log_\epsilon(1+\epsilon^{-z})=c\ln(1+e^{-z/c}) \tag{22}$$

where $c=1/\ln\epsilon=\log_\epsilon e$, would appear to be too complicated to be implemented. Figure 3 plots $f(z)$ against z for $c=1$. We can see that $f(z)$ quickly

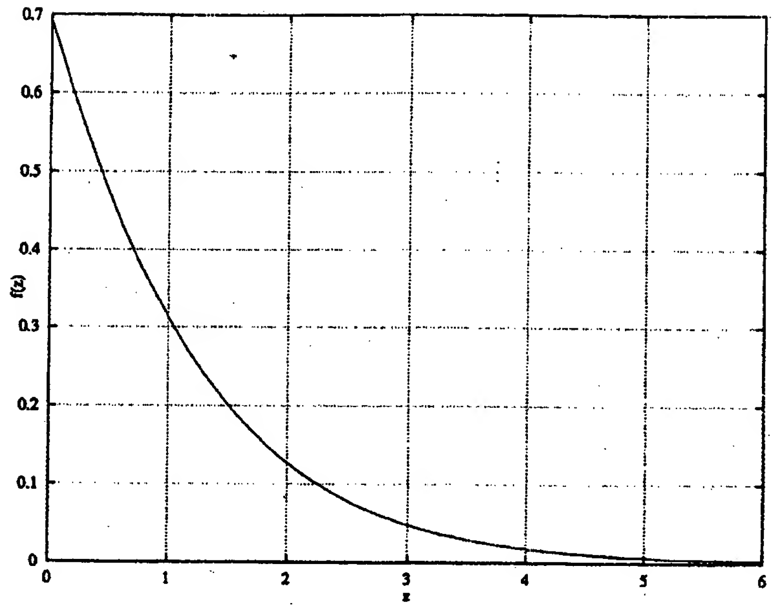

Figure 3. Plot of $f(z)$ versus z for $c = 1$

decays to zero and has a maximum value of $c \ln 2 \approx 0{\cdot}693c$ (for $z \geq 0$). Thus $f(z)$ can be easily implemented in a small look-up table. A range of look-up tables can be implemented to cover various values of c.

If we let

$$L_k = -\log_e \lambda_k \tag{23a}$$

$$A_k^m = -\log_e \alpha_k^m \tag{23b}$$

$$B_k^m = -\log_e \beta_k^m \tag{23c}$$

$$D_k^{i,m} = -\log_e \delta_k^{i,m} \tag{23d}$$

the MAP algorithm becomes

$$L_k = \mathop{\mathrm{E}}_{m=0}^{2^\nu-1} A_k^m + D_k^{0,m} + B_{k+1}^{f(0,m)}$$

$$- \mathop{\mathrm{E}}_{m=0}^{2^\nu-1} A_k^m + D_k^{1,m} + B_{k+1}^{f(1,m)} \tag{24}$$

$$A_k^m = \mathop{\mathrm{E}}_{j=0}^{1} A_{k-1}^{b(j,m)} + D_{k-1}^{j,b(j,m)} \tag{25}$$

$$B_k^m = \mathop{\mathrm{E}}_{j=0}^{1} D_k^{j,m} + B_{k+1}^{f(j,m)} \tag{26}$$

where $\mathrm{E}_{j=0}^{l-1} \alpha^j = \alpha^0 \mathrm{E}\, \alpha^1 \mathrm{E} \cdots \mathrm{E}\, \alpha^{l-1}$. The branch metrics are

$$\begin{aligned} D_k^{i,m} &= -\log_e \kappa_k - \log_e \zeta_k^i - A(x_k i + y_k c^{i,m}) \\ &= -\log_e \kappa_k - \log_e \zeta_k^0 - i \log_e(\zeta_k^1/\zeta_k^0) \\ &\quad - A(x_k i + y_k c^{i,m}) \\ &= -K_k - (z_k + Ax_k)i - Ay_k c^{i,m} \end{aligned} \tag{27}$$

where K_k is a constant, $A = (2/\sigma^2)\log_e e = L_c c$ and $z_k = -\log_e \zeta_k$ is the log-APrP. To perform renormalization, we let K_k be equal to the smallest previous state metric. We can think of A as the no-noise amplitude of our demodulated and quantized signal, e.g. +1 could be equivalent to $A = 7 = 111_2$. Note that we can arbitrarily vary A and L_c to determine c and thus the values in a look-up table for (22). Alternatively, given L_c and a look-up table for a value of c, we can vary A. We can re-express (20) as

$$L_k = z_k + Ax_k + z'_k \tag{28}$$

where $z'_k = -\log_e \zeta'_k$ is the extrinsic information from the log-MAP decoder.

2.3. *The sub-MAP Decoding Algorithm*

If we let $f(z) = 0$, then (24)–(26) become

$$\mathrm{L}_k = \min_m (A_k^m + D_k^{0,m} + B_{k+1}^{f(0,m)})$$

$$- \min_m (A_k^m + D_k^{1,m} + B_{k+1}^{f(1,m)}) \tag{29}$$

$$\mathrm{A}_k^m = \min (A_{k-1}^{b(0,m)} + D_{k-1}^{0,b(0,m)}, A_{k-1}^{b(1,m)}$$

$$+ D_{k-1}^{1,b(1,m)}) \tag{30}$$

$$B_k^m = \min (D_k^{0,m} + B_{k+1}^{f(0,m)}, D_k^{1,m} + B_{k+1}^{f(1,m)}) \tag{31}$$

This suboptimal algorithm (which we shall call sub-MAP) has the advantage that it is independent of σ^2. Again, this algorithm was first derived for the ISI channel[25,26] and then later applied to the coding channel.[19,30] It has the same suboptimal hard-decison performance as the Viterbi algorithm.[19] Note that the calculation of the forward state metrics is exactly the same as the state metric (SM) calculation for the Viterbi algorithm. However, unlike the Viterbi algorithm, the SMs also need to be calculated in the reverse direction and the likelihood ratio determined.

3. IMPLEMENTING THE LOG-MAP ALGORITHM

As can be seen from (24)–(27), there are four major sections in a log-MAP decoder, These are the forward state metric calculator (FSMC), the reverse state metric calculator (RSMC), the log likelihood ratio calculator (LLRC) and the branch metric calculator (BMC). To minimize the decoder gate count, it was decided to have a serial implementation; that is, the SMs are computed one at a time. This is similar to previous serial Viterbi and trellis decoders constructed by the author.[31, 32] Since there are 2^{ν} states, this implied that it would take at least 2^{ν} decoder clock (CLK) cycles to decode each bit. Thus the CLK frequency had to be at least 2^{ν} times greater than the data clock (DCLK) frequency.

To avoid problems with high-speed clocks, it was decided to limit the CLK speed to 10 MHz. Also, the Xilinx XC3100A[33] series programmable logic was chosen owing to its low cost and relatively high speed.

3.1. *Branch Metric Calculator*

From Reference 34 the BMs are calculated as follows:

$$D_k^{i,m} = |z_k + Ax_k|(i \oplus u(z_k + Ax_k)) + |Ay_k|(c^{i,m} \oplus u(Ay_k)) - K_k \qquad (32)$$

where $i \oplus j$ is the modulo-2 sum of i and j and $u(w)$ is the unit step function. As before, K_k is equal to the minimum previous state metric. In two's complement notation, $u(w)$ corresponds to the logical inverse of the most significant or sign bit of w. It can be shown that

$$|w|(j \oplus u(w)) = -wj + (w + |w|)/2 \qquad (33)$$

The $-wj$ term in (33) corresponds directly to the terms in (27) with $w = z_k + Ax_k$ and $j = i$ or $w = Ay_k$ and $j = c^{i,m}$. The additional terms are constants dependent on k only and can be absorbed in K_k. We can see that when the BM in (32) is added to its state metric, the resulting value will always be greater than or equal to zero.

An important consideration is the value of the signal amplitude A. The optimum value of A can vary depending on the number of quantization bits q and the noise variance σ^2.[35] Figure 4 illustrates a model of the demodulator for the received signal x_k (as defined in (3)). A model of the branch metric calculator is also shown (we have assumed that $z_k = 0$ in this case). We assume that x_k is multiplied by some unknown fixed positive voltage V. The demodulator has an automatic gain control (AGC) circuit that effectively normalizes the input signal to its mean absolute value, i.e.

$$\begin{aligned} E[|Vx_k|] &= VE[|2d_k - 1 + p_k|] \\ &= V\left[\sigma\sqrt{\frac{2}{\pi}}\exp\left(-\frac{1}{2\sigma^2}\right) + 1 - 2Q\left(\frac{1}{\sigma}\right)\right] \\ &= V\text{mag}(\sigma) \end{aligned} \qquad (34)$$

Note that for high SNR (and low σ), $\text{mag}(\sigma) \approx 1$. However, for low SNR (and high σ) we have $\text{mag}(\sigma) \approx \sigma\sqrt{2/\pi} \approx 0{\cdot}798\sigma$. This is very important when trying to estimate the noise variance. A variance estimator that assumes $\text{mag}(\sigma) = 1$ will work correctly only for high SNR.

For turbo codes where a low SNR is expected, a more complicated method is required to determine V and σ. We can see from (34) that we have two unknowns and one equation. To solve for V and σ, we need another equation which can be obtained by estimating the square of the received signal:

$$E[(Vx_k)^2] = V^2(1 + \sigma^2) \qquad (35)$$

Figure 5 plots $\text{mag}(\sigma)$ and $\text{rms}(\sigma) = \sqrt{1 + \sigma^2}$. We can see that for low SNR, $\text{rms}(\sigma) \approx \sigma$. Owing to the complexity of (34), there does not seem to be a simple direct solution of the two equations. The estimation of (34) and (35) can be performed digitally. By quantizing (34) and (35) into, say, 8 bits each, a 64K × 8 look-up table can be used to output precomputed σ values quantized to 8 bits each. Alternatively, the ratio of the square of (34) with (35) will give a single equation as a function of σ.[36] A smaller 256 × 8 look-up table can then be used to output σ.

The value of A compared with the dynamic range of an analogue-to-digital (A/D) converter can greatly affect the decoded BER[35] (especially when the number of quantization levels is small). We shall assume that there are $2^q - 1$ quantization regions with a central 'dead zone'; that is, a quantized '0' ranges from $-0{\cdot}5$ to $0{\cdot}5$. The largest quantized value is $2^{q-1} - 1$ and ranges from $2^{q-1} - 1{\cdot}5$ to infinity. As shown in our demodulator model in Figure 4, we shall assume that the demodulator scales the input by C before A/D conversion. In Reference 35, computer simulations of the Viterbi algorithm showed that C should be less than one. Also, as the SNR is decreased, the value of A relative to the maximum quantized output should decrease as well. From Figure 4 we have that the 'optimum' value of A is

$$A = \frac{C(2^{q-1} - 1)}{\text{mag}(\sigma)} \qquad (36)$$

Analysing the simulations in Reference 35, we found that $C = 0{\cdot}65$ should give near-optimum perform-

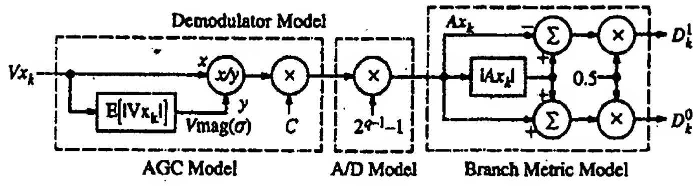

Figure 4. Demodulator and branch metric model

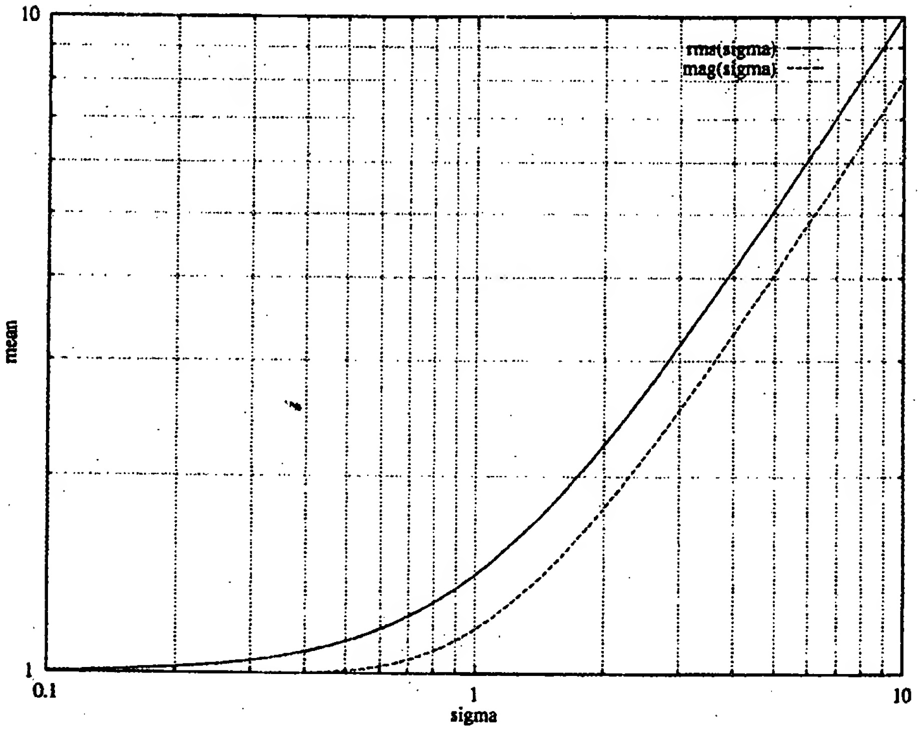

Figure 5. Absolute mean and root mean square of received signal

ance. However, for low σ it may be necessary to reduce C (and thus A) to keep the look-up tables for the E operand at a reasonable size.

For our BMC it was decided to have $q = 6$ bit quantization as a compromise between having good performance and decoder complexity. Each MAP decoder could be programmed for either rate 1/2, 1/3 or 1/4 operation. For rate 1/4 mode and $z_k = 0$ this implied that the maximum BM value is $n(2^{q-1} - 1) = 124$, where n is the number of coded bits. In turbo decoding mode, z_k can range from −128 to +127, which can greatly increase the maximum BM. Owing to the limitation of the number of bits to represent each SM; the maximum BM was therefore limited to 127.

One BM is calculated each CLK cycle with a two-CLK-cycle pipeline delay (one cycle to determine the symbol and another cycle to perform the calculation). The BM is then passed onto the FSMC and an SRAM for storage. The BMs stored in the SRAM are then read out in reverse order for the RSMC. Since 2^ν BMs are calculated for N DCLK cycles, the total storage space required in $N2^\nu$. Although there are only 2^n BMs, we chose to simply store and then later retrieve the previously calculated BMs for the RSMC.

Ideally, for the $N = 2^{16}$ turbo code in Reference 6 the required memory storage is one megabyte (MB). This was too large and too expensive to implement and so it was decided to limit the storage space to 64K. For $\nu = 4$ this implies that $N = 2^{12} = 4096$. A simplistic storage technique is to write the new data into one 64K RAM and read the old data from another 64K RAM. This requires a total of 128K of RAM. We can halve the amount of RAM by using the circuit shown in Figure 6.

When CLK is high, the previously stored BM is read out from the RAM and then latched on the falling edge of CLK. Simultaneously, a new BM is written into the RAM using the same address when CLK goes low. Thus, in one clock cycle we perform a read followed by a write. Since the RAM address is inverted every N DCLK cycles, the BMs are read out in reverse order. A control signal to the $\overline{\mathrm{CE}}$

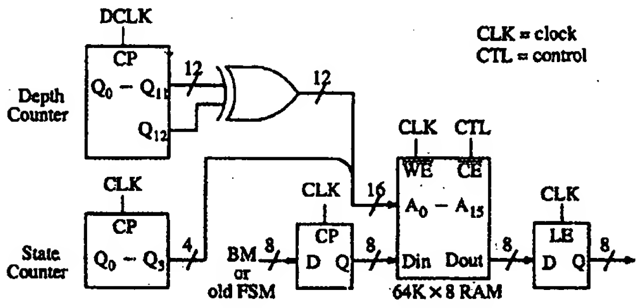

Figure 6. Branch and state metric storage for block MAP decoder

input of the RAM is used to enable the storage of the new SMs at the correct time. In our design, two 35 ns 64K × 4 separate I/O SRAMs were used.

Our design could be programmed from four to 512 states, with a corresponding change in N from 16,384 to 128. However, to limit the decoding delay and storage requirements for the turbo decoder, the maximum block size for four and eight states was reduced to 4096.

3.2. *State Metric Calculators*

The architecture of the FSMC and RSMC are very similar. A 35 ns 1K × 8 dual-port RAM is used to retrieve the old SMs and store the new SMs. With 8 bit precision the SMs can range from zero to 255. A further increase in precision would have greatly increased the complexity of the decoder. Thus it was decided that the SMs would be represented by 8 bits.

At the end of the previous block the initial SMs are stored into one side of the RAM. For the FSMs the SM for state zero is set to zero and the other states are set to 255 (the closest value to infinity). This corresponds to the sequence starting in state zero. For the RSMs, if the final state is unknown, all the initial SMs are set to zero, otherwise they are initialized in the same way as the FSMs.

To determine the read and write addresses for the SMs, we need to examine the implementation of a rate $1/n$ systematic encoder. Berrou *et al.*[6] showed that a rate 1/2 systematic encoder can be implemented using a shift register as shown in Figure 7. We have that $S_k = (s_k^0, s_k^1, s_k^2, \ldots, s_k^{\nu-1})$ corresponds to the current encoder state, $G_i = (g_i^0, g_i^1, g_i^2, \ldots, g_i^{\nu})$ corresponds to the encoder code and $g_i^j, s_k^j \in \{0,1\}$ for $0 \le i \le n-1$. We assume that $g_i^0 = g_i^{\nu} = 1$ for $0 \le i \le n-1$, since this ensures that the free Hamming distance leaving and entering a state is at least $2n$.

For the forward SMs we need to determine $b(d_{k-1}, S_k)$ from (25). If we let

$$b(d_{k-1}, S_k) = S_{k-1} = (s_k^1, s_k^2, \ldots, s_k^{\nu-1}, s_k^0) \qquad (37)$$

then

$$S_k = (d_{k-1} \oplus s_k^0 \oplus s_k^*, s_k^1, s_k^2, \ldots, s_k^{\nu-1}) \qquad (38)$$

where

$$s_k^* = \sum_{j=1}^{\nu-1} g_0^j s_k^j \bmod 2 \qquad (39)$$

We can see that by reading two SMs we can generate two new SMs. Figure 8 gives a partial trellis for a $\nu = 4$ code. Thus in the next data clock (DCLK) cycle the previously stored SMs are read

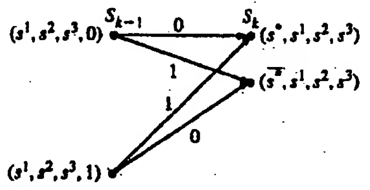

Figure 8. Subtrellis for $\nu = 4$, rate $1/n$ systematic convolutional code

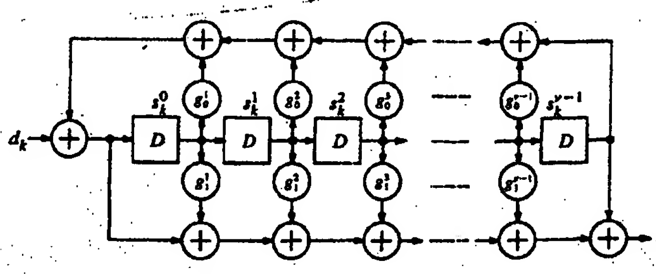

Figure 7. Rate 1/2 systematic convolutional encoder

out one at a time using the following forward read address (ignoring the subscript k):

$$R_f = (s^1, s^2, \ldots, s^{\nu-1}, s^0) \qquad (40)$$

A serial-to-parallel operation is performed and the two SMs are stored in a register for two CLK cycles. In the first CLK cycle, BM^0 is added to the first SM and BM^1 is added to the second SM to form the first new SM. In the next CLK cycle the BMs are reversed to form the second new SM. After a delay from reading and calculating the new SMs (equal to five CLK cycles), the new SMs are written into the dual-port RAM with the following address (ignoring the subscript k):

$$W_f = (s^0 \oplus s^*, s^1, s^2, \ldots, s^{\nu-1}) \qquad (41)$$

From (26) we need to determine $f(d_k,S_k)$ for the reverse direction. If we let

$$f(d_k,S_k) = S_{k+1} = (d_k \oplus s_k^0 \oplus s_k^*, s_k^1, s_k^2, \ldots, s_k^{\nu-1}) \qquad (42)$$

then

$$S_k = (s_k^1, s_k^2, \ldots, s_k^{\nu-1}, s_k^0) \qquad (43)$$

Thus in a similar way to the FSMC the read and write addresses for the RSMC are

$$R_r = (s^0 \oplus s^*, s^1, s^2, \ldots, s^{\nu-1}) \qquad (44)$$

$$W_r = (s^1, s^2, \ldots, s^{\nu-1}, s^0) \qquad (45)$$

Since there is a delay in calculating the new SMs, we cannot use the read/$\overline{\text{write}}$ technique as used for storing the BMs. With a 1K × 8 DP-RAM this implied the maximum number of states is 512 (half the memory size). The minimum number of states is four owing to the restriction that $g_f^0 = g_f^\nu = 1$.

Since the forward SMs are used in the LLRC, they also need to be stored and read out in reverse order. A circuit very similar to the BM storage circuit is used to perform this task. The old forward SMs that are read from the DP-RAM are the values that are stored in two 64K × 4 SRAMs.

An important part of the SM calculator is the implementation of the adders and the E operand. Figure 9 illustrates how the BMs are added to the SMs (similar to that in Reference 37). We see that the previous minimum SM is subtracted from the BMs before being added to the SMs. The output of the BM subtraction circuit has a two's complement output (we limit the output so that the range is from −128 to +127). The BMs are then added to the SMs (just as in the Viterbi algorithm). If the BM is positive, the SM limiting circuit is enabled. However, if the BM is negative, the limiting circuit is disabled to allow normal addition to occur. Since the BM can never be more negative than the smallest SM, the resulting SM will always be positive.

Figure 10 illustrates the E operand circuit. As shown in (21), we need to find the minimum of the two path metrics (PMs) as well as the absolute difference. The carry-out of a subtraction circuit (with carry-in set to zero) is used to select the smallest path metric. This carry-out is also used to invert the output of another subtraction circuit to give the absolute difference. When PM^0 is greater than PM^1, the output of the subtractor will give the correct positive output when the carry-in is equal to one. However, if PM^0 is less than or equal to PM^1, then the output will be negative. Normally, we would have to invert the output and then add one. However, by setting the carry-in to zero, we avoid the additional adder circuit. This is why the carry-out of the first subtractor goes into the carry-in of the second subtractor.

A circuit that uses a comparator, two multiplexers and a subtractor can also be used to implement the minimum and absolute difference functions.[29] In XC3100A logic this would require at least 20 configurable logic blocks (CLBs) to implement using 8 bit arithmetic, compared with 16 CLBs for our design (since the XNORs can be absorbed into the second comparator).

In order to keep the SMs positive, we add $c \ln 2$ to (21). Thus, if the absolute difference is equal to zero, we add zero to the minimum, otherwise we add some small value. As determined previously, the maximum value of $f(z)$ is $c \ln 2$. For a rate 1/7 code, $E_b/N_0 = -0{\cdot}5$ dB and 6 bit quantization we have $\sigma^2 = 3{\cdot}93$, and using (36), $A = 11{\cdot}3$ (which we round to 11). Thus $c = \sigma^2 A/2 = 21{\cdot}6$ and the maximum value of $f(z)$ is 14·99 (15 after rounding). Thus only 4 bits are required to represent $f(z)$.

The smallest non-zero value of $f(z)$ that results in zero after quantization is 0·5. Solving for $f(z) = 0{\cdot}5$, we have

$$z = -c \ln[\exp(0{\cdot}5/c) - 1] \qquad (46)$$

For the above conditions we have $z = 81{\cdot}2$. Since $f(z)$ decreases monotonically with z, all values of $f(z)$ for $z \geq 81{\cdot}2$ will be less than 0·5 and so will be quantized to zero. Therefore, by limiting values above 81 (the quantized value of 81·2) from the absolute difference circuit to 81, we only require an 81 × 4 look-up table to implement $f(z)$. In our design we limited the maximum address space to 63 in order to reduce the design complexity. This implied from (46) that the largest value of c is 17·835 (giving $z = 63{\cdot}5$). Thus, if the optimum value of A causes c to be more than 17·835, we reduce A to give us the maximum c. For the above example this would be $A = 9{\cdot}08$ (or $A = 9$ after rounding).

Figure 10 shows how the absolute difference output is limited to 6 bits and used to address a 64 × 4 look-up table to determine $f(z) + c\ln 2$. The table look-up output is then added to the minimum circuit

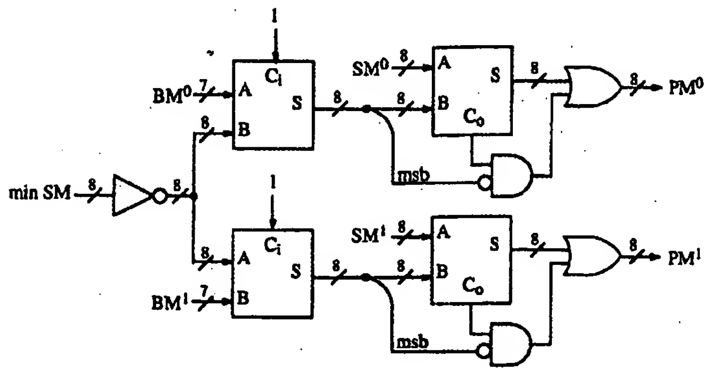

Figure 9. BM and SM adder circuit

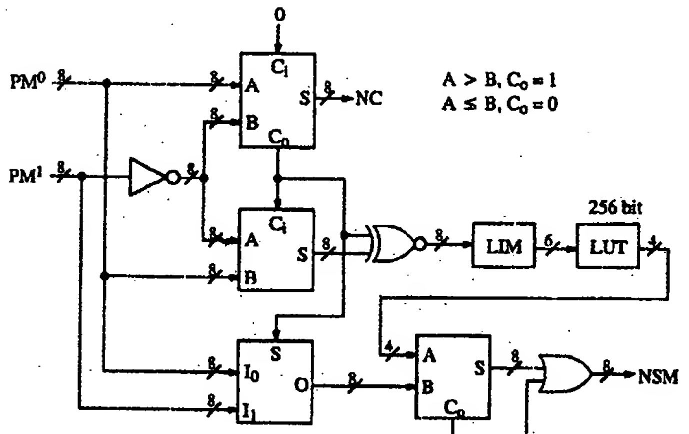

Figure 10. Add–compare–select E circuit

output. The adder output is limited to 8 bits as shown. Just as in a Viterbi decoder, limiting the outputs of the adders will cause some degradation in performance. However since it is the larger and thus less likely path metrics that are limited, this degradation will be very small.

3.3. *Log Likelihood Ratio Calculator*

For the computation of λ_k^i (EM^i) we add the reverse path metric for bit i (RPM^i) to the corresponding forward SM that is read from the SRAM. This is shown in Figure 11. The summation gives a 9 bit result which is not limited. A most significant sign bit is also added owing to the effect of $f(z)$ (in this case we cannot add $c$ln2 because we are E summing more than one term). The register is initialized to its maximum value (+511) to start the E summation (or 'eccumulation'). We could have used a multiplexer to initialize the EM^i register to the first summation. However, to reduce complexity, a simple limiting circuit was used. Since on the first E summation +511 is usually much greater than the first metric, the EM^i register would be correctly initialized most of the time.

We then find the minimum and absolute differ-

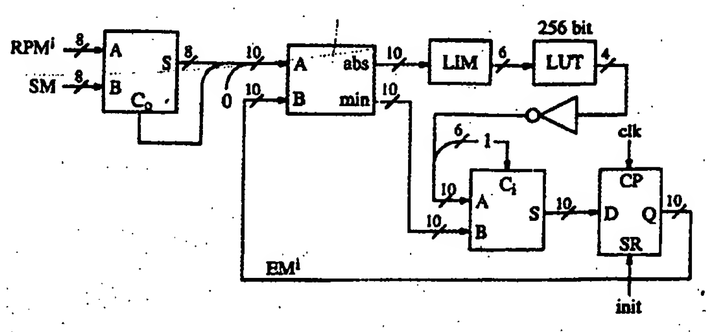

Figure 11. Eccumulator circuit

ence between the current register output and the current metric (the summation of RPM^i and SM). We then perform the standard look-up operation, in this case subtracting the look-up result from the minimum. To decrease the delay between the register output and its input, we also register the outputs of the abs-min circuit. This forced us to insert a multiplexer after the first adder (the other input was the EM^i output) to allow the correct EM^i to be calculated owing to pipeline delay.

After the EM's have been finally calculated (which is done in parallel), we subtract EM^1 from EM^0 and limit the output to +127 or −128 if necessary to give an 8 bit result. An additional three clock cycles are required to perform the final calculation. Since the LLRs are calculated with the reverse SMs, the LLRs are produced in reverse order. Thus for a normal MAP decoder the output needs to be reversed in time in blocks of N.

3.4. *Decoder Performance*

Since up to two clock cycles are required for the decoder to start, two for the BMC, five for the SMCs and three for the LLRC, a total of $2^\nu + 12$ clock cycles are required. All the logic was implemented in XC3100A-5 gate arrays which allowed a clock speed of $f_c = 10$ MHz. The SRAMs were 35 ns in speed. Also, ν bits were used to terminate the trellis, which slightly reduces the speed of the decoder. The decoder speed is given by

$$f_d = \begin{cases} \dfrac{(2^{12}-\nu)f_c}{2^{12}(2^\nu+12)}, & 2 \le \nu \le 4 \\ \dfrac{(2^{16-\nu}-\nu)f_c}{2^{16-\nu}(2^\nu+12)}, & 5 \le \nu \le 9 \end{cases} \tag{47}$$

The slowest decoder speed is for $\nu = 9$ at 17·7 kbit/s and the fastest decoder speed is for $\nu = 2$ at 624·7 kbit/s. For $\nu = 4$ the speed is 356·8 kbit/s.

The SMCs and LLRC were implemented in one XC3190A-5 (7500-gate equivalent). The BMC was implemented in one XC3142A-5 (3700-gate equivalent), while two XC3130A-5s (2700-gate equivalent) were used to implement the control logic and address generation (as well as some additional functions for the turbo decoder). The encoder was implemented using an XC3142A-5. Both the encoder and decoder can be programmed with any code with $g^0_i = g^\nu_i = 1$ through a series of DIP switches.

To test the decoder performance, AWGN was generated on a PC using a C++ program. The parallel port on the back of the PC was used to transmit 8 bit quantized noise samples to the decoder. An adder circuit in the encoder Xilinx chip adds the noise to the encoded signal. The adder circuit produces a 6 bit quantized result with a dead zone and 63 quantization regions. Various noise generators were used, starting with one that had a period of $8{\cdot}4 \times 10^6$ for the MAP decoder tests, $2{\cdot}1 \times 10^9$ for the rate 1/3 turbo decoder tests and $2{\cdot}3 \times 10^{18}$ for the rate 1/7 turbo decoder tests. This last decoder used the 'standard' Lehmer uniform random number generator with multiplier 16807 and modulus $2^{31} - 1$,[38] together with the Box–Muller algorithm from Reference 39 (pp. 216–217) to generate 2^{23} 8 bit quantized random numbers which are stored in the RAM (after inputting A and σ). The dual 32 bit speed uniform random number generator from Reference 40 was then used to randomly select the numbers from the RAM and send them to the encoder.

Figure 12 gives the performance of a rate 1/4, 512-state systematic code with code polynomials $g_0 = 1753$, $g_1 = 1547$, $g_2 = 1345$ and $g_3 = 1151$ taken from Reference 41. The performance of the hardware log-MAP decoder is plotted, along with the performance of the hardware sub-MAP decoder and a software block Viterbi decoder. The signal amplitude was fixed to $A = 7$ to avoid limiting the SMs too much. This is less than the 'optimum' values for 6 bit quantization, but with only 8 bit state metrics, SM limiting becomes more of a factor. At low BER we can see that the implementation loss is only 0·05 dB. Note that at low BER the ideal performances of the Viterbi and MAP algorithms are almost identical, which allows us to make a comparison. At high BER the MAP decoder gains about 0·5 dB over ideal Viterbi.

Also plotted on the graph is the sub-MAP decoder performance. This is where the E operand is simplified to the min function. This was done by setting the look-up tables in the MAP decoder to zero. The signal amplitude was also set to $A = 7$. As can be seen, at low BER the sub-MAP performance is almost identical to that of the MAP decoder. However, the sub-MAP decoder always performed worse than the ideal Viterbi, losing about 0·3 dB at high BER.

Figure 13 shows the performance for a systematic rate 1/2, 16-state code with polynomials $g_0 = 37$ and $g_1 = 21$ from Reference 6. Since the rate loss from the tail bits is very small, we could compare our decoder results with a software MAP decoder from Reference 42. The hardware MAP decoder closely follows the computer simulation at high BER, losing 0·08 dB at low BER. The sub-MAP decoder loses about 0·5 dB at high BER and closely follows the MAP decoder at low BER.

4. TURBO CODING AND DECODING

The basic turbo encoder consists of two or more parallel concatenated systematic convolutional encoders separated by an interleaver of size N_i. Figure 14 shows the general implementation of a rate 1/3 turbo encoder from two constituent rate 1/2 encoders. The two coded outputs can be punctured in order to obtain higher rates. The INT block is the interleaver and the DEL block is a delay circuit with a delay equal to the MAP decoder delay. The DEL circuit

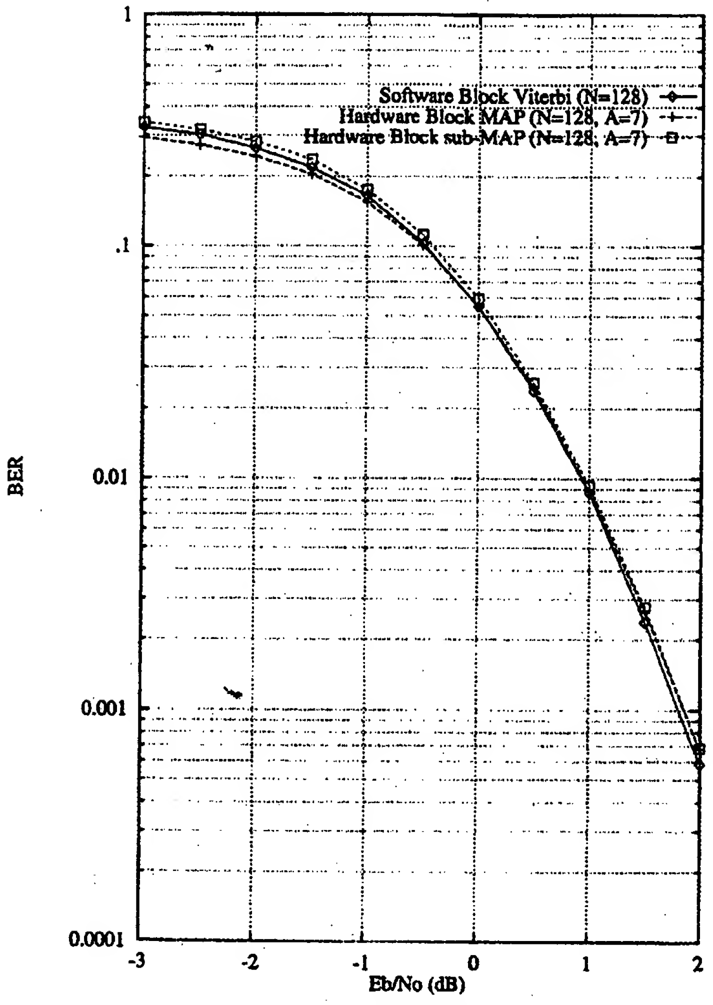

Figure 12. Systematic rate 1/4, 512-state BER performance

is required since the decoder for the interleaved data has to wait for the first decoder to output its data. Thus the received data have to be appropriately delayed. Instead of the decoder delaying the received interleaved data, this delay is performed within the encoder, simplifying the decoder operation.

Figure 15 illustrates the basic decoding block in an iterative turbo decoder. In the first iteration we do not need to add the APrP for d_k to Ax_k. Since $d_k = 0$ or 1 is equally likely, we have that $z_k^2 = 0$ (a superscript 2 is used here as explained later) is added to Ax_k. We then decode the symbols from the first encoder. The output from the first MAP decoder is $\lambda_j^1 = Ax_j + z_j^1$, where the superscripts indicate which MAP decoder was used, and $j = k - D_d$, where D_d is the delay of the MAP decoder. These data are then interleaved to match the interleaved symbols from the second encoder.

The $Ax_j + z_j^1$ from the first MAP decoder is then fed into the second MAP decoder. In this case we have let the extrinsic information from the first MAP decoder become the APrP for the second MAP decoder. One can think of the first MAP decoder improving the SNR for Ax_j, which effectively results in a lower BER for d_k. The purpose of the interleaver is to randomize the 'burst' errors that are characteristic of MAP and Viterbi decoders. The larger the interleaver, the more randomized are the bursts of errors.

With an improved Ax_j the second MAP decoder is able to correct even more errors. Its output is

$$\lambda_i^2 = z_i^1 + Ax_i + z_i^2 \qquad (48)$$

where z_i^2 is the extrinsic information from the second MAP decoder and the subscript i is used to indicate the interleaved and delayed time index. We subtract $z_i^1 + Ax_i$ from λ_i^2 to obtain z_i^2, which is then deinterleaved and passed onto another iteration as $z_{k-\Delta}^2$ (where Δ is the total delay of the iteration). Just like in the second MAP decoder, this extrinsic information becomes the APrP for the first MAP decoder in the next iteration. The deinterleaver serves to randomize the burst errors from the second MAP decoder. Note that if we include z_i^1 and z_i^2, the deinterleaver would produce a delayed z_j^1 with 'bursty'

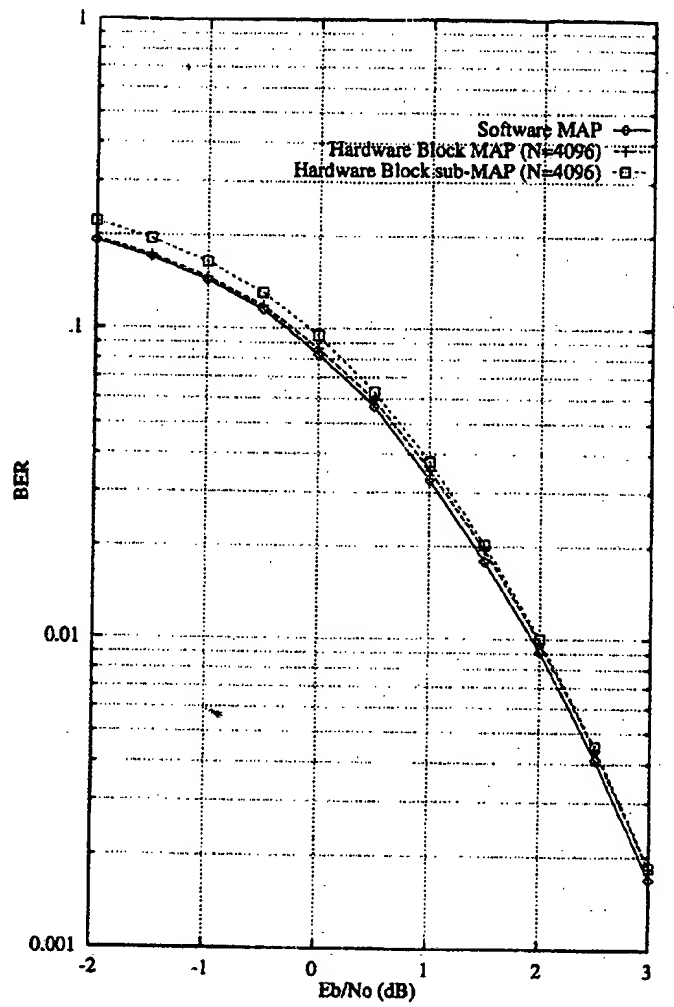

Figure 13. Systematic rate 1/2, 16-state (37,21) BER performance

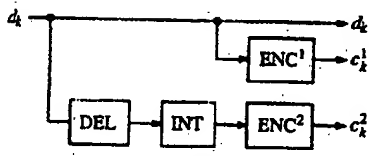

Figure 14. Rate 1/3 turbo encoder

errors. Thus feeding these bursty errors into the first MAP decoder will result in very poor performance from the decoder.

In the next iteration the first MAP decoder output is

$$\lambda_j^1 = z_j^2 + Ax_j + z_j^1 \tag{49}$$

In this case we subtract z_j^2 from λ_j^1 to give the

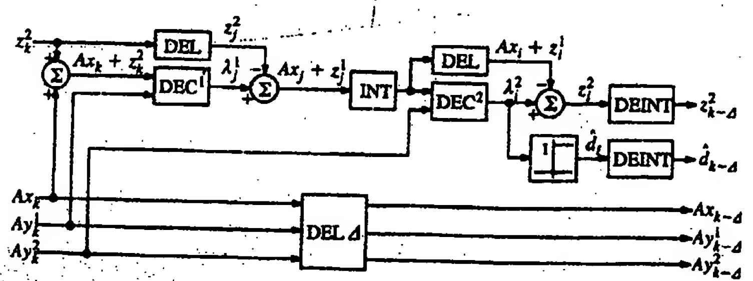

Figure 15. Interative turbo decoder

desired $Ax_j + z_j^1$, which is then fed into the second MAP decoder as before. Again, we do not want to include z_j^2, since after interleaving it would be bursty again.

Obviously, in the first stage of decoding we want to obtain the lowest BER possible. However, at low SNRs a code's performance may be opposite to what is expected. An important observation is that the more powerful a code is at low BERs, the *worse* it performs at low SNR and high BERs.[42] Thus we do not want to choose too complex a code, as iterative decoding will give poor performance. However, we also do not want to choose too weak a code (in terms of performance at low BER), as the latter stages of decoding will not be powerful enough to correct any further errors. This was first noticed in Reference 6, where a 16-state code was found to be optimal for a rate 1/2 turbo decoder.

4.1. *Decoder implementation*

The additional circuits required for a turbo decoder are the delays for z_j^2 and $Ax_i + z_i^1$, the interleavers and deinterleavers, the delay for $Ax_k, Ay_k^1, \ldots, Ay_k^{n-1}$, an adder and two subtractors. For the last stage of decoding the deinterlever can be used to deinterleave the second MAP decoder output with the addition of a multiplexer. With a rate 1/4 MAP decoder a rate 1/7 turbo decoder could be constructed. To reduce the number of inputs, the data were received serially, one symbol at a time. Thus BPSK modulation could be directly used, although with some modifications QPSK could also be used.

Each MAP decoder has a maximum delay of 4096 bits and, as shown later, each interleaver has a maximum delay of 65,536 bits. Thus the total delay for the seven symbols is $7 \times 2 \times (4096 + 65{,}536) = 974{,}848$. Thus six $1\text{M} \times 1$ SRAMs were used to delay the received data. The MAP decoder was initially designed to have a 16K block size and thus a 16K delay (for $\nu = 2$). With this extra delay, six $256\text{K} \times 1$ SRAMs were included in the design, but are no longer necessary. Also, two $16\text{K} \times 4$ separate I/O SRAMs were used for each of the z_j^2 and $Ax_i + z_i^1$ delays, although two $4\text{K} \times 4$ SRAMs would be sufficient.

Since the interleaver and deinterleaver architectures are similar, we will only discuss the implementation of the interleaver. For the interleaver we have used the same read/$\overline{\text{write}}$ technique as used by the delay and reversing circuits elsewhere in the design (thus giving a delay equal to the interleaver block size). The maximum interleaver size is the same as in Reference 6, i.e. $N_i = 64\text{K}$. Thus only two $64\text{K} \times 4$ separate I/O SRAMs are required to implement the interleaver. However, the interleaver address generator circuit is a little more complex and is shown in Figure 16. At start-up the SEL line goes high and the counter output is stored into the SRAM. The SRAM output is then read out and latched by the D-FFs. Using this address, the data are sequentially stored into the interleaver SRAM. At the same time, SEL goes low and the interleaver address is read from the EPROM and stored into the SRAM, to be read out in the next interleaver block. The process repeats, interleaving the previous set of addresses.

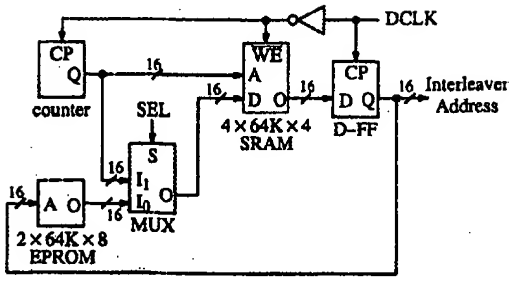

Figure 16. Interleaver address generator

Note that only one interleaver address generator (IAG) is implemented. Thus only one set of EPROMs needs to be programmed with any interleaver that is desired. The time reversal of the MAP decoder output can also be incorporated into the interleaver EPROM, reducing the delay and complexity of the decoder. Owing to the two MAP decoder delays (equal to 8K), the interleaver address between each iteration also has to be delayed. We used two $8\text{K} \times 8$ SRAMs to perform this task. Since $8\text{K} \times 8$ SRAMs have common I/O, the circuit in Figure 17 was used to separate the I/O and allow the read/$\overline{\text{write}}$ technique to be used.

A disadvantage of the above scheme is that if an error occurs in the address generator, error propagation occurs and the decoder will have to be reset. However, in the many days of testing we performed on our codec, the decoder never had to be reset owing to the interleaver failing or any other part of the decoder failing.

The first encoder sequence starts and ends in state zero through the use of a ν-bit tail. Interleaving is performed on all the information and tail bits. The second encoder sequence is also made to start in state zero. However, to keep the same decoder architecture for the second MAP decoder, the N bits are not forced to end in state zero. This implies that the final state is unknown. The second MAP decoder takes this into account by initializing all the reverse state metrics to zero.

Figure 18 shows a photograph of the encoder and IAGs. To the left-hand side are seven DIP switches

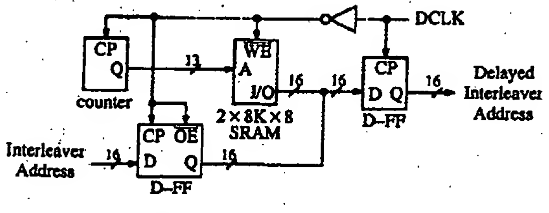

Figure 17. Interleaver address delay

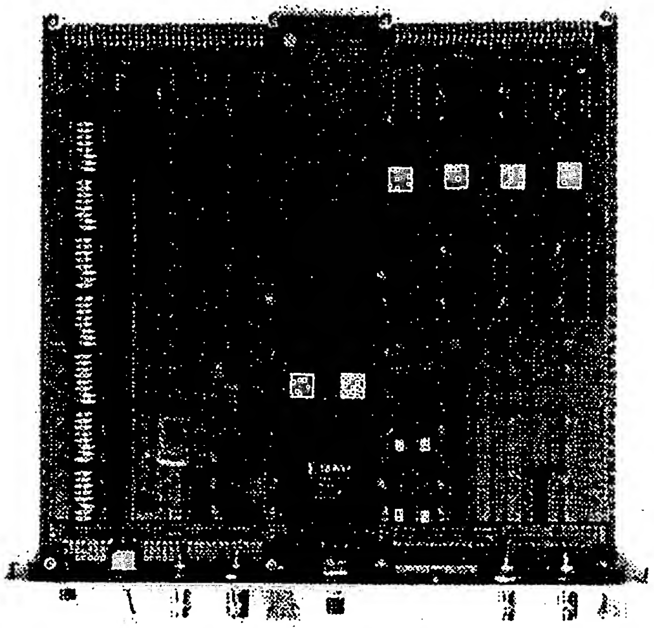

Figure 18. Turbo encoder and interleaver address generator

used to programme the code polynomials for both the encoder and decoder. The bottom Xilinx chip performs the encoder function. The middle Xilinx chip is the address counter and multiplexer for the IAG. Figure 19 shows a decoder iteration. The first decoder prototype was implemented using speedwire, which allowed any design corrections to be easily made. This prototype was then implemented on a printed circuit board. From the top, the Xilinx chips are the control logic and address generator, MAP decoder 1, MAP decoder 2, data delay address generator and miscellaneous logic (left) and branch metric calculator (right). The large chips to the left and right of the Xilinx chips are the $1K \times 8$ dual-port SRAMs used to store the new and old state metrics (SMs). The left chips store the reverse SMs and the right chips store the forward SMs. To the left of the control logic chip are the $64K \times 4$ SRAMs where we store the branch metrics to be used in calculating the reverse SMs. The four $64K \times 4$ SRAMs to the right of the control logic chip are used to store the forward SMs to be

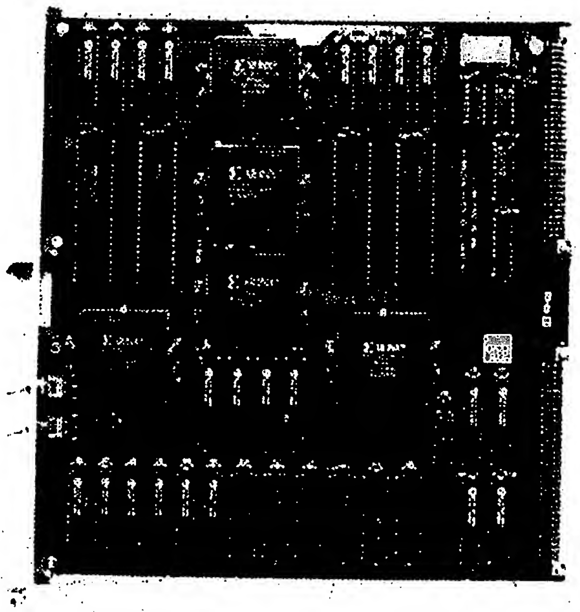

Figure 19. Turbo decoder iteration

used in calculating the log likelihood ratio. Between the bottom two Xilinx chips are the 16K × 4 SRAMs for delaying the input and producing the extrinsic information. The 12 SRAMs at the bottom left of the board are for delaying the n 6 bit inputs (six 256K × 1 and six 1M × 1 SRAMs). The four 64K × 4 SRAMs at the bottom right-hand corner perform the deinterleaving and interleaving of the data.

Figure 20 is a photograph of the completed codec. A 6U high 48 cm rack is used which contains, from left to right, the encoder/interface card, turbo/MAP decoder card 1, interleaver address delay card 1, turbo/MAP decoder cards 2–5, interleaver address delay card 2 and turbo/MAP decoder cards 6 and 7. An additional 11 turbo/MAP decoder cards can fit within the rack to give a total of 18 iterations. Turbo/MAP decoder 7 has its switches in the up position, indicating that the decoder is set up for seven iterations. Each iteration can also output the first decoder output, using a 16K × 4 SRAM from the second MAP decoder to reverse the data in time.

Note that our current decoder implementation is not able to automatically synchronize to a received signal. Instead, a signal from the encoder was used to synchronize the decoder. Future modifications may include a synchronization word in the encoded block to allow synchronization.

4.2. *Turbo Decoder Performance*

The first tests were made using a rate 1/3 turbo code using identical rate 1/2 16-state codes with polynomials $g_0 = 31$ and $g_1 = 33$ (in octal).[43] These code polynomials were optimized for rate 1/3 turbo codes and were found to perform better than the codes from Reference 6 (which were optimized for a rate 1/2 turbo code). The value of A was set to 15 for all E_b/N_0, which is close to the optimum values. An $S = 31$,[44] 65,536 bit interleaver was used ($S = 31$ implies that any two consecutive bits are separated by at least 31 other bits after interleaving). The actual rate is reduced by 4092/4096 = 1023/1024 owing to the MAP block size being only 4096 bits, with 4 bits used as the tail. The 'inner' code is terminated to state zero, while the 'outer' code is not terminated. Shannon capacity at this rate is at an $E_b/N_0 = -0{\cdot}55$ dB (the capacity at this rate with QPSK modulation is $-0{\cdot}49$ dB).

In Figure 21 we plot BER versus E_b/N_0 for 6·5 and seven iterations (a half-iteration is the output of the first MAP decoder). We see that 10^{-5} and 10^{-6} are achieved at 0·32 and 0·38 dB respectively. These are 0·87 and 0·93 dB away from rate 1/3 capacity for BERs of 10^{-5} and 10^{-6} respectively. Unfortunately, 10^{-7} is close to an E_b/N_0 of 1·0 dB owing to the BER flattening above 0·4 dB. We also see that above 0·4 dB the BER from 6·5 iterations performs better than that from seven iterations. This may be due to the second MAP decoder being unterminated or perhaps an effect of other non-linearities in the decoder.

Also note how the performance shallows between 0·35 and 0·4 dB. There appears to be a sudden change in slope. As can be seen, it is not an error floor, since the BER does decrease with increasing E_b/N_0. The cause of this sudden change in slope is the small free distance of turbo codes.[45] This free

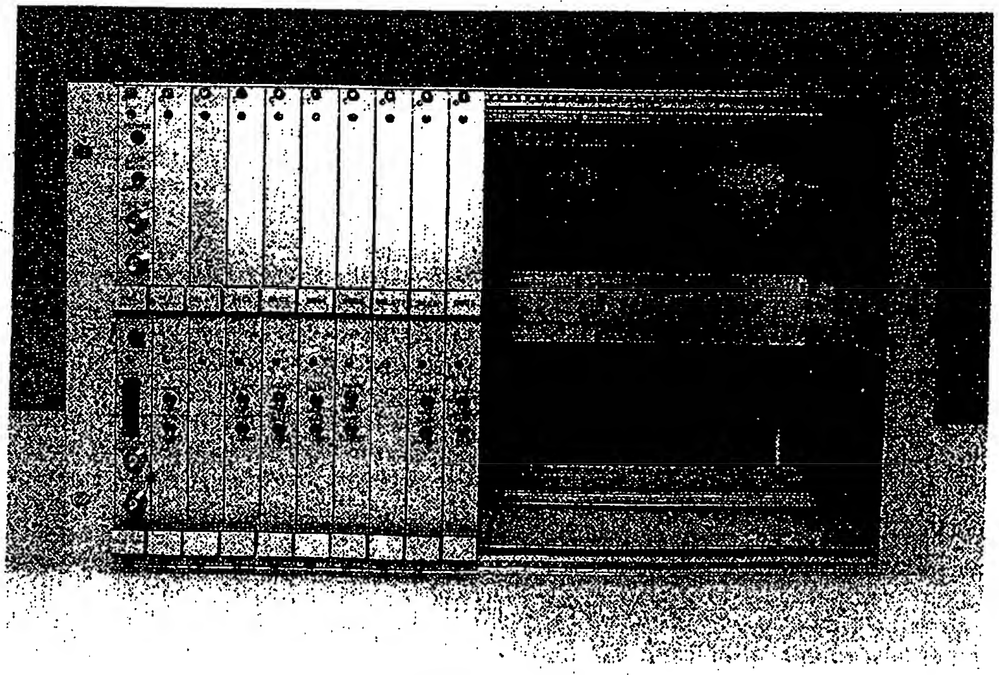
Figure 20. Turbo codec

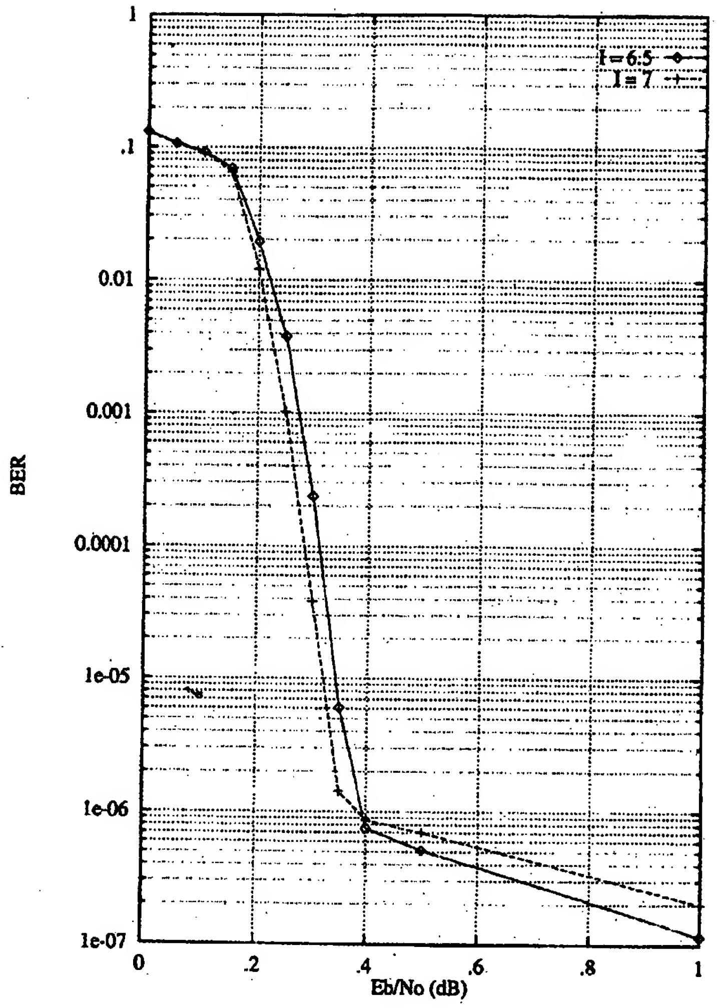

Figure 21. Rate 1/3, 16-state (31,33), $N_i = 65{,}536$ turbo decoder performance versus E_b/N_0

distance has a spectral component much less than one, which greatly reduces its error probability (in fact, this reduction is inversely proportional to N_i[45]). However, since the free distance term has a shallow slope, this slope will appear at relatively high E_b/N_0.

Computer simulations for this scheme have not been previously performed. Thus we make a comparison with the rate 1/3, 16-state, $N_i = 16{,}384$ scheme in Reference 46. With 11 iterations it achieved an E_b/N_0 of 0·24 dB at a BER of 10^{-5}. This is 0·08 dB better than our scheme, which has seven iterations and an interleaver that is four times larger.

The first MAP decoder had mostly double-bit error outputs at low BER. This is predicted in Reference 45, where the use of systematic convolutional codes leads to error sequences of information weight two (corresponding to the error bit causing the sequence to leave the path and the error bit causing the sequence to return to the path). This is very important for turbo codes, since it greatly increases their performance over the use of non-systematic enoders (which have single-bit error patterns).[45] Also, systematic encoders perform slightly better over non-systematic encoders at low SNR,[42] which is important in iterative decoding.

The output from the second MAP decoder at low BERs was mostly in single-bit errors, indicating that the unterminated states could be causing a problem. Since there are $N_i/N = 16$ MAP blocks in each interleaver block, the effect of all these unterminated states could induce these single-bit errors.

It was found that if the subtractor outputs in Figure 15 were limited to have a range from −128 to +127, the decoder would perform very poorly. When the extrinsic information is added to the received noisy sample (which ranges from −31 to +31), errors could be introduced owing to non-linearities in the design. For example, say we receive +31 and add the extrinsic information +127 to it. The output of the first MAP decoder will be limited to +127. When we subtract the extrinsic information, the information fed into the second MAP decoder will then be zero! The results shown in Figure 15 had the subtractor outputs limited from −64 to +63 to avoid this problem.

In Figure 22 we plot the BER against the number

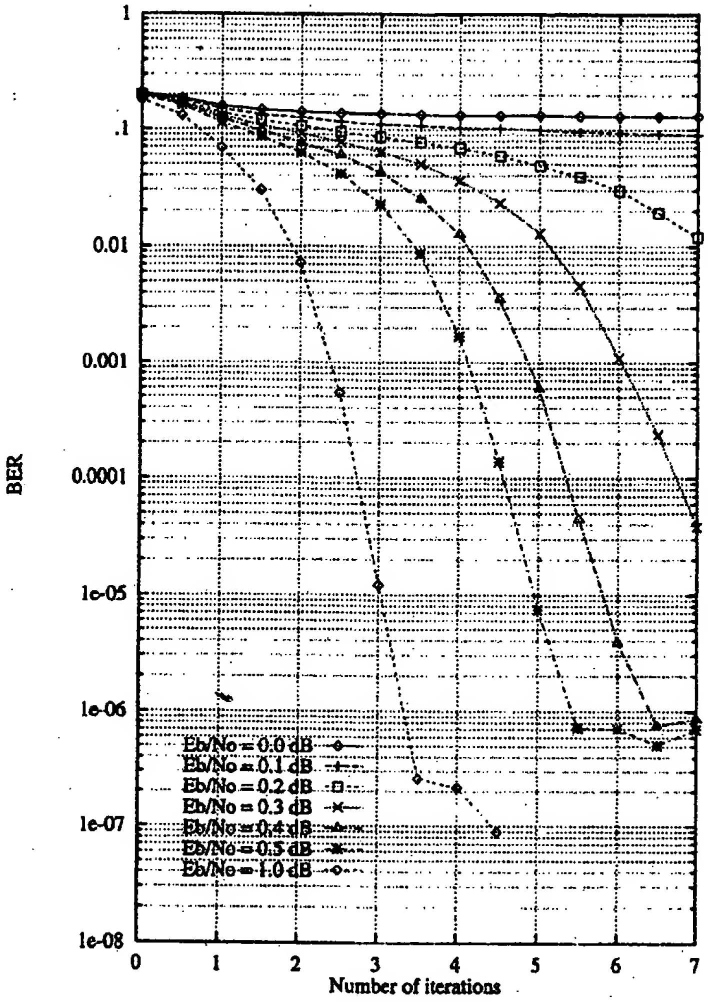

Figure 22. Rate 1/3, 16-state (31,33), $N_I = 65{,}536$ turbo decoder performance versus I

of iterations for $E_b/N_0 = 0{\cdot}0$, 0·1, 0·2, 0·3, 0·4, 0·5 and 1·0 dB. Of interest is how the BER suddenly flattens after quickly decreasing for 0·4, 0·5 and 1·0 dB. Another point of interest is the potential of more iterations. The 0·2 dB curve indicates that further improvement is possible and the 0·1 dB curve might be able to reach low BERs as well.

Figure 23 shows the performance of our rate 1/7 16-state turbo decoder with code polynomials $g_0 = 23$, $g_1 = 35$, $g_2 = 27$ and $g_3 = 37$ from Reference 46. To obtain better performance, the extrinsic information subtractor outputs were limited from −96 to +95. We set $A = 7$ to avoid limiting the state metrics too greatly. The 64K interleaver from Reference 24 was tried but was found to give inferior performance to our randomly generated interleaver (although it must be noted that this interleaver was designed for a rate 1/2 turbo code).

Values of E_b/N_0 of −0·30, −0·27 and −0·19 dB are achieved for BERs of 10^{-5}, 10^{-6} and 10^{-7} respectively. Note that the 10^{-7} BER is achieved with 6·5 iterations. Shannon capacity at this rate is at −1·12 dB, from which we are 0·82 dB away at a BER of 10^{-5}. With more iterations we expect to reduce this gap by 0·1–0·2 dB.

5. CONTINUOUS DECODING AND SYNCHRONIZATION

A way of improving the decoding design is to use a continuous MAP decoder. A continuous decoder does not need a tail to be added to the sequence and only needs to synchronize to the n coded symbols. We will give a description of an efficient implementation of a continuous MAP decoder along with a synchronization technique that can be used for a turbo decoder.

5.1. *Continuous Decoding*

A continuous decoding algorithm for the MAP algorithm was first presented for the ISI channel in Reference 47 (with a complexity exponentially proportional to the decoder delay). A simpler algorithm for continuous decoding of convolutional codes was first described in Reference 48 and later in

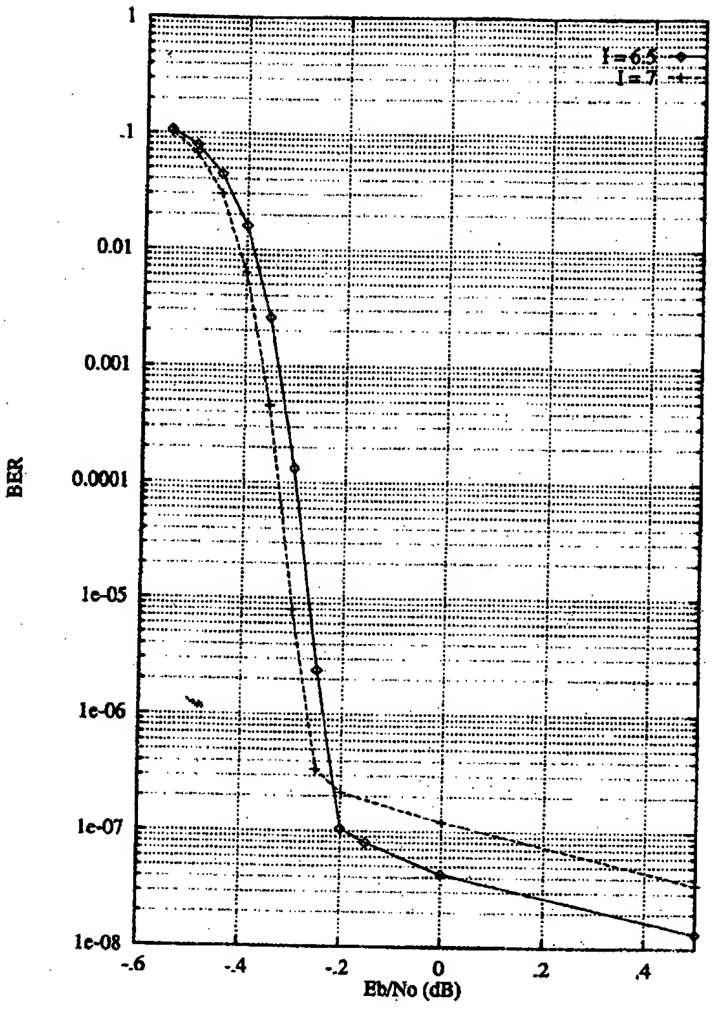

Figure 23. Rate 1/7, 16-state $N_i = 65{,}536$ turbo decoder performance versus E_b/N_0

References 29, 49 and 50. A similar algorithm for the ISI channel is given in Reference 51. The n received data symbols (R_k) are stored from time k to $k+L-1$. They are then read out in reverse order from time $k+L-1$ to k and β^m_{k+1} determined recursively (starting with $\beta^m_{k+L} = 1$ for all m). The forward SMs α^m_k are also calculated as normal using the R_k that have been delayed by $2L$. Using α^m_k, $\delta^{i,m}_k$ and β^m_{k+1}, λ_k is determined as normal. We then increment k by one and repeat the whole algorithm. By making L sufficiently large, the β^m_{k+1} that is determined will be very close to the true β^m_{k+1} had β^m_{k+L} been known precisely. We can see that this algorithm is computationally intensive, since each β^m_k is calculated L times instead of only once.

A computationally simpler algorithm was first mentioned in Reference 18 and later in more detail in References 52–54. As for the previous algorithm, we store R_k into a RAM of size nL. The reverse SMs are then calculated (starting with $\beta^m_{k+L} = 1$ for all m). However, the reverse SMs continue to be calculated from time k to $k-L+1$ using the R_k from the reversing RAM. After the R_k have been delayed by $2L$ (using another RAM of size $2nL$), the forward SMs are calculated and stored in a RAM of size $2^\nu L$. The time-reversed forward SMs are combined with the last L reverse SMs to give a time-reversed LLR output. We can see that only half the reverse SMs that are calculated are used. Thus two reverse SM calculators are used in pipeline.

Figure 24 shows how R_k is stored and read for use by the reverse SMCs. In this case we have assumed that $L = 64$. The first RAM is used to time reverse R_k in blocks of L and the second RAM is used to delay the reversed R_k by $2L$. Two multiplexers are then used to alternatively select the RAM outputs for use by the two RSMCs.

Using the read/$\overline{\text{write}}$ technique, the total amount of RAM that is required is $5L \times nq + L2^\nu \times 8$, where we have assumed 8 bit SMs and LLR. An additional $2^\nu \times 8$ bits would be required if the output had to be reversed in time. This is still considerably less complex than having L RSMCs as in the previous algorithm. If the decoder is to be used in a turbo

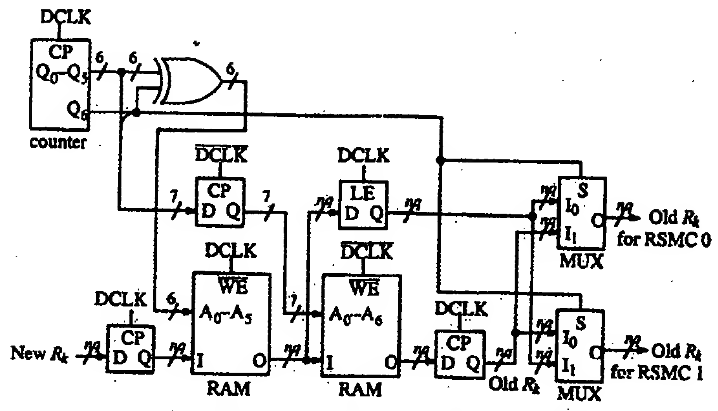

Figure 24. Storage of R_k for reverse SM calculators

decoder, this reversal can be performed within the interleaver. The total delay is $4L$ (with the LLR reverser) or $3L$ (without the LLR reverser).

5.2. *A Synchronization Technique for Turbo Decoders*

In a turbo decoder an important consideration is being able to synchronize to the interleaved data of depth N_i. One could insert synchronization words, but this increases the complexity and decreases the bandwidth efficiency. The first MAP decoder needs only to synchronize to n possible states. This can be done by monitoring the average amplitude of L_k. When the decoder is out of synchronization, the average value of $|L_k|$ will be lower than expected. This is due to the decoder not receiving a valid code sequence and producing an unreliable decoded sequence.

By accumulating $|L_k|$ for a certain length of time and comparing it with a threshold, the decoder can make a reliable decision as to whether it is synchronized. If the decoder is not synchronized, it tries the next state and then waits for a decoder delay (in order for the decoder to produce stable data) before it starts monitoring $|L_k|$ again. This process continues until synchronization is achieved.

The average synchronization time (t_s) depends on the decoder delay (t_d), the time to average $|L_k|$ (t_a) and the number of synchronization states (n_s). Since we can randomly start in any state (taking on average $(n_s-1)/2$ attempts before synchronization is achieved), we have that

$$t_s = (t_d + t_a)(n_s - 1)/2 \qquad (50)$$

The worst-case synchronization time is only twice the average synchronization time. For a turbo decoder we have

$$t_s = (t_d + t_a)(n - 1)/2 + (N_i + t_d + t_a)(N_i - 1)/2 \qquad (51)$$

The first part of (51) corresponds to the first MAP decoder and the second part to the second MAP decoder. For large N_i the synchronization time will be dominated by the second MAP decoder.

For example, if $t_d = 3L = 192$, $t_a = 64$, $n = 2$ and $n_i = 65{,}536$, then $t_s = 2{,}155{,}839{,}488$ bits! At 2·048 Mbit/s it would thus take on average about 17·5 min to synchronize. Once synchronized, one should increase t_a so that the synchronization circuit does not indicate a false alarm with very high probability.

A way of reducing the overall synchronization time is to evenly distribute the synchronization time between the two decoders. We could do this by adding modulo-2 a random binary sequence of length L_1 to the parity of the first individual code. This forces the inner decoder to synchronize to nL_1 states.

We also have that $L_1L_2 = N_i$ (to keep the overall number of synchronization states the same). The outer decoder now needs only to synchronize to L_2 states. The average synchronization time is then

$$t_s = (t_d + t_a)(nL_1 - 1)/2 + (N_i + t_d + t_a)(L_2 - 1)/2 \qquad (52)$$

One can then try various values of L_1 and L_2 to minimize (52). Assuming a real-valued L_2, the optimum value of L_1 is (taking the differential of (52) and setting it to zero)

$$L_1 = \sqrt{\frac{N_i(N_i + t_d + t_a)}{n(t_d + t_a)}} \qquad (53)$$

For our example we thus have $L_1 = 2901·96$ and $L_2 = 22·58$. Since L_1 and L_2 must be integers, we let $L_1 = 2849$, $L_2 = 23$ and $N_i = 65{,}527 = 2^{16} - 9$. From (52) we have $t_s = 1{,}452{,}829$ bits, a nearly 1500 times reduction compared with the original scheme. At 2·048 Mbit/s this corresponds to an average synchronization time of 0·709 s.

Alternatively, we could let $L_1 = 4096$ and $L_2 = 16$, which only slightly increases t_s to 1,541,888 bits (a delay of 0·753 s at 2·048 Mbit/s). The combination of $L_1 = 2048$ and $L_2 = 32$ is only marginally slower at 1,543,936 bits or 0·754 s.

6. CONCLUSIONS

The original turbo coding paper by Berrou *et al.*[6] presented a coding scheme that came within 0·7 dB of Shannon capacity. The MAP decoding algorithm that was presented was much too complicated to implement practically. We have rederived the MAP algorithm to present it in as simple a form as possible. By taking the logarithm of the MAP algorithm, a realizable implementation could be achieved at high data rates.

The turbo decoder that we have constructed has indeed been able to verify the amazing performance presented in Reference 6. We were able to come within 0·8 dB of capacity with only seven iterations. With up to 18 iterations we can expect to reduce this amount by 0·1–0·2 dB. Thus we have been able to demonstrate that near-Shannon performance can be achieved at high data rates.

We used a block-type MAP algorithm which requires a lot of memory for its implementation. An efficient implementation of a continuous decoder has been presented. Continuous decoders have the advantage of being less complex, are easier to synchronize, have a much smaller delay and have slightly better performance.

A synchronization technique for turbo decoders which takes advantage of the low delay of continuous decoders has also been presented. It is shown that even for large interleaver sizes, automatic synchronization can be achieved in a relatively small time interval.

ACKNOWLEDGEMENTS

The author would like to thank Riccardo Macri for providing the AWGN-generating program.

This work was performed under the sponsorship of the Australian Research Council and the University of South Australia. The material in this paper was partially presented at the International Symposium on Information Theory, Whistler, BC, Canada, September 1995 and the International Symposium on Information Theory and its Applications, Victoria, BC, Canada, September 1996.

APPENDIX. GLOSSARY OF ABBREVIATIONS

ACS	add–compare–select
AGC	automatic gain control
APoP	*a posteriori* probability
APrP	*a priori* probability
AWGN	additive white Gaussian noise
BER	bit error ratio
BM	branch metric
BMC	branch metric calculator
BPSK	binary phase shift keying
CE	clock enable
CLB	configurable logic block
CLK	clock
CP	clock input
CTL	control
D-FF	data flip-flop
DCLK	data clock
DEC	decoder
DEINT	deinterleaver
DEL	delay
DIP	dual inline package
DPRAM	dual-port random access memory
EM	eccumalator metric
ENC	encoder
EPROM	erasable programmable read-only memory
FSM	forward state metric
FSMC	forward state metric calculator
I/O	input/output
IAG	interleaver address generator
INT	interleaver
ISI	intersymbol interference
LLR	log likelihood ratio
LLRC	log likelihood ratio calculator
MA	multiply–add
MAP	maximum *a posteriori*
MB	megabyte
MUX	multiplexer
OE	output enable
PC	personal computer
PM	path metric
QPSK	quadrature phase shift keying
RAM	random access memory
RS	Reed–Solomon
RSM	reverse state metric
RSMC	reverse state metric calculator
SEL	select
SISO	soft-in soft-out
SM	state metric
SMC	state metric calculator
SNR	signal-to-noise ratio
SRAM	static random access memory
SOVA	soft-output Viterbi algorithm
WE	write enable
XNOR	exclusive negative OR

REFERENCES

1. C. E. Shannon, 'A mathematical theory of communications', *Bell Syst. Tech. J.*, **27**, 379–423, 623–656 (1948).
2. P. E. McIlree, 'Channel capacity calculations for *M*-ary *N*-dimensional signal sets', *MEng Thesis*, University of South Australia, 1995.
3. J. P. Odenwalder, 'Optimal decoding of convolutional codes', *PhD Dissertation*, University of California, Los Angeles, CA, 1970.
4. E. C. Posner, L. L. Rauch and B. D. Madsen, 'Voyager mission telecommunication firsts', *IEEE Commun. Mag.*, **28**, 22–27 (1990).
5. S. Dolinar and M. Belongie, 'Enhanced decoding for the Galileo low-gain antenna mission: Viterbi redecoding with four decoding stages', *JPL TDA Prog. Rep.*, **42**(121), 96–109 (1995).
6. C. Berrou, A. Glavieux and P. Thitimajshima, 'Near Shannon limit error-correcting coding and decoding: turbo-codes', *IEEE Int. Conf. on Communications*, Geneva, May 1993, pp. 1064–1070.

7. L. Bahl, J. Cocke, F. Jelinek and J. Raviv, 'Optimal decoding of linear codes for minimizing symbol error rate', *IEEE Trans. Info. Theory*, **IT-20**, 284–287 (1974).
8. R. G. Gallager, 'Low density parity check codes', *IRE Trans. Info. Theory*, **IT-8**, 21–28 (1962).
9. A. J. Viterbi, 'Error bounds for convolutional codes and an asymptotically optimum decoding algorithm', *IEEE Trans. Info. Theory*, **IT-13**, 260–269 (1967).
10. S. A. Barbulescu, W. Farrell, P. Gray and M. Rice, 'Bandwidth efficient turbo coding for high speed mobile satellite communications', *Int. Symp. on Turbo Codes*, Brest, September 1997, pp. 119–126.
11. G. Battail, 'Pondération des symbols décodés par l'algorithm de Viterbi', *Ann. Télécommun.*, **42**, 1/1–1/8 (1987).
12. J. Hagenauer and P. Hoeher, 'A Viterbi algorithm with soft-decision outputs and its applications', *GLOBECOM '89*, Dallas TX, November 1989, pp. 47.1.1–47 1.7.
13. J. Huber and A. Rüppel, 'Zuverlässigkeitsabschätzung für die Ausgangssymbole von Trellis decodern', *AEÜ (Electron. Commun.)*, **44**, 8–21 (1990).
14. C. Berrou, P. Adde, E. Angui and S. Faudeil, 'A low complexity soft-output Viterbi decoder architecture', *IEEE Int. Conf. on Communications*, Geneva, May 1993, pp. 737–740.
15. J. Hagenauer, P. Robertson and L. Papke, 'Iterative (turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms', *ITG Tagung, Codierung für Quelle, Kanal und Übertragung*, Frankfurt, October 1994, pp. 21–29.
16. C. Berrou and G. Lochon, 'CAS 5093 turbo-code codec', *Data Sheet*, COMATLAS, Chateaubourg, 1993.
17. M. Jézéquel, C. Berrou, J. R. Inisan and Y. Sichez, 'Test of a turbo-encoder/decoder', *Turbo Coding Seminar*, Lund, August 1996, pp. 35–42.
18. S. S. Pietrobon and S. A. Barbulescu, 'A simplification of the modified Bahl decoding algorithm for systematic convolutional codes', *Int. Symp. on Information Theory and Its Applications*, Sydney, November 1994, pp. 1073–1077.
19. P. Robertson, E. Villebrun and P. Hoeher, 'A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain', *ICC '95* Seattle, WA, June 1995, pp. 1009–1013.
20. R. W. Chang and J. C. Hancock, 'On receiver structures for channels having memory', *IEEE Trans. Info. Theory*, **IT-12**, 463–468 (1966).
21. L. Bahl, J. Cocke, F. Jelinek and J. Raviv, 'Optimal decoding of linear codes for minimizing symbol error rate', *IEEE Int. Symp. on Information Theory*, Asilomar, CA, May 1972, p. 90.
22. P. L. McAdam, L. R. Welch and C. L. Weber, 'M.A.P. bit decoding of convolutional codes', *IEEE Int. Symp. on Information Theory*, Asilomar, CA, May 1972, p. 91.
23. P. Robertson, 'Improving decoder and code structure of parallel concatenated recursive systematic (turbo) codes', *Int. Conf. on Universal Personal Communications*, San Diego, CA, September–October 1994, pp. 183–187.
24. C. Berrou and A. Glavieux, 'Near optimum error correcting coding and decoding: turbo-codes', *IEEE Trans. Commun.*, COM-44, 1261–1271 (1996).
25. J. Erfanian, S. Pasupathy and G. Gulak, 'Reduced complexity symbol detectors with parallel structures for ISI channels', *IEEE Trans. Commun.*, **COM-42**, 1661–1671 (1994).
26. W. Koch and A. Baier, 'Optimum and sub-optimum detection of coded data disturbed by time varying intersymbol interference', *GLOBECOM '90*, San Diego, CA, December 1990, pp. 1679–1684.
27. S. S. Pietrobon, 'Implementation and performance of a serial MAP decoder for use in an iterative turbo decoder', *IEEE Int. Symp. on Information Theory*, Whistler, BC, September 1995, p. 471.
28. E. Villebrun, 'Turbo-decoding with close-to-optimal MAP algorithms', *Diploma Thesis*, TU Munich, 1994.
29. S. Benedetto, G. Montorsi, D. Divsalar and F. Pollara, 'Soft-output decoding algorithms in iterative decoding of turbo codes', *JPL TDA Prog. Rep.* 42(124), 63–87 (1996).
30. J. Petersen, 'Implementierungsaspekte zur Symbol-by-Symbol MAP Decodieurung von Faltungscodes', *ITG Tagung, Codierung für Quelle, Kanal und Übertragung*, Frankfurt, October 1994, pp. 41–48.
31. S. S. Pietrobon, 'Discrete implementation of a NASA planetary standard Viterbi decoder', *IREE Int. Electronics Conv. Exhib.*, Sydney, September 1987, pp. 249–252.
32. S. S. Pietrobon and D. J. Costello Jr., 'A bandwidth efficient coding scheme for the Hubble Space Telescope', *J. Elec. Electron. Engng., Aust.* **13**, 275–282 (1993).
33. *The Programmable Logic Data Book*, Xilinx, San Jose, CA, 1996.
34. O. M. Collins, 'The subtleties and intricacies of building a constraint length 15 convolutional decoder', *IEEE Trans. Commun.*, **COM-40**, 1810–1819 (1992).
35. I. M. Onyszchuk, K.-M. Cheung and O. Collins, 'Quantization loss in convolutional decoding', *IEEE Trans. Commun.*, **COM-41**, 261–265 (1993).
36. T. A. Summers and S. G. Wilson, 'SNR mismatch and online estimation in turbo decoding', *IEEE Trans. Commun.*, **46**, 421–423 (1998).
37. S. S Pietrobon, J. J. Kasparian and P. K. Gray, 'A multi-D trellis decoder for a 155 Mbit/s concatenated codec', *Int. J. Satell. Commun*, **12**, 539–553 (1994).
38. S. K. Park and K. W. Miller, 'Random number generators: good ones are hard to find', *Commun. ACM*, **31**, 1192–1201 (1988).
39. W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipies in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1988.
40. P. L'Ecuyer, 'Efficient and portable combined random number generators', *Commun. ACM*, **31**, 742–749, 774 (1988).
41. P. J. Lee, 'Further results on rate 1/*N* convolutional code constructions with minimum required SNR criterion', *IEEE Trans. Commun.*, **COM-34**, 395–399 (1986).
42. J. Y. Couleaud, 'High gain coding schemes for space communications', *ENSICA Final Year Report*, University of South Australia, 1995.
43. S. Benedetto and G. Montorsi, 'Design of parallel concatenated convolutional codes', *IEEE Trans. Commun.*, **COM-44**, 591–600 (1996).
44. D. Divsalar and F. Pollara, 'Multiple turbo codes for deep-space communications', *JPL TDA Prog. Rep.* 42(121), 66–77 (1995).
45. S. Benedetto and G. Montorsi, 'Unveiling turbo codes: some results on parallel concatenated coding schemes', *IEEE Trans Info. Theory*, **IT-42**, 409–428 (1996).
46 D. Divsalar and F. Pollara, 'On the design of turbo codes', *JPL TDA Prog. Rep.* **42**(123), 99–121 (1995).
47. K. Abend and B. D. Fritchman, 'Statistical detection for communication channels with intersymbol interference', *Proc. IEEE*, **58**, 779–785 (1970).
48. L.-N. Lee, 'Real-time minimal-bit-error probability decoding of convolutional codes', *IEEE Trans. Commun.*, **COM-22**, 146–151 (1974).
49. K.-H. Tzou and J. G. Dunham, 'Sliding block decoding of convolutional codes', *IEEE Trans. Commun.*, **COM-29**, 1401–1403 (1981).
50. X. Wang and S. B. Wicker, 'A soft-output decoding algorithm for concatenated codes', *IEEE Trans. Info. Theory*, **IT-42**, 543–553 (1996).
51. Y. Li, B. Vucetic and Y. Sato, 'Optimum soft-output detection for channels with intersymbol interference', *IEEE Trans. Info. Theory*, **IT-41**, 704–713 (1995).
52. S. A. Barbulescu, 'Iterative decoding of turbo codes and other concatenated codes', *PhD Dissertation*, University of South Australia, 1996, pp. 23–24.
53. S. S. Pietrobon, 'Efficient implementation of continuous MAP decoders and a synchronisation technique for turbo decoders', *Int. Symp. on Information Theory and Its Applications*, Victoria, BC, September 1996, pp. 586–589.
54. A. J. Viterbi, 'An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes', *IEEE J. Select. Areas Commun.*, Submitted.

Authors' biography:

Steven S. Pietrobon was born in Naracoorte, South Australia on 5 October 1963. He received BEng and MEng degrees in electronic engineering from the South Australian Institute of Technology (SAIT, now University of South Australia), Adelaide, South Australia in 1986 and 1989 respectively and a PhD degree in electrical engineering from the University of Notre Dame, Notre Dame, Indiana, USA in 1991. He received a SAIT Medal for outstanding academic achievement for his BEng degree. In 1991 he joined the Australian Space Centre for Signal Processing, University of South Australia as a Research Fellow. In 1993 he was awarded a three-year Australian Postdoctoral Research Fellowship at the University of South Australia. In 1997 he started his own business, Small World Communications, developing error control decoders for programmable gate arrays. Dr Pietrobon also became Adjunct Research Fellow at the Institute for Telecommunications Research, University of South Australia in 1997. Since 1996 he has been a Coding Theory and Techniques Editor for the *IEEE Transactions on Communications*. His research interests include convolutional, trellis and turbo coding and implementation of Viterbi, trellis, MAP and turbo decoders.